FILE TRANSFER PROTOCOL (FTP)

J.Postel , J.Reynolds
Network Working Group , Request for Comments: 959 (October 1985)

Status of this Memo

This memo is the official specification of the File Transfer Protocol (FTP). Distribution of this memo is unlimited.

The following new optional commands are included in this edition of the specification:

> CDUP (Change to Parent Directory), SMNT (Structure Mount), STOU (Store Unique), RMD (Remove Directory), MKD (Make Directory), PWD (Print Directory), and SYST (System).

Note that this specification is compatible with the previous edition.

1 INTRODUCTION

The objectives of FTP are 1) to promote sharing of files (computer programs and/or data), 2) to encourage indirect or implicit (via programs) use of remote computers, 3) to shield a user from variations in file storage systems among hosts, and 4) to transfer data reliably and efficiently. FTP, though usable directly by a user at a terminal, is designed mainly for use by programs.

The attempt in this specification is to satisfy the diverse needs of users of maxi-hosts, mini-hosts, personal workstations, and TACs, with a simple, and easily implemented protocol design.

This paper assumes knowledge of the Transmission Control Protocol (TCP) [2] and the Telnet Protocol [3]. These documents are contained in the ARPA-Internet protocol handbook [1].

2 OVERVIEW

In this section, the history, the terminology, and the FTP model are discussed. The terms defined in this section are only those that have special significance in FTP. Some of the terminology is very specific to the FTP model; some readers may wish to turn to the section on the FTP model while reviewing the terminology.

2.1 HISTORY

FTP has had a long evolution over the years. Appendix III is a chronological compilation of Request for Comments documents relating to FTP. These include the first proposed file transfer mechanisms in 1971 that were developed for implementation on hosts at M.I.T. (RFC 114), plus comments and discussion in RFC 141.

RFC 172 provided a user-level oriented protocol for file transfer between host computers (including terminal IMPs). A revision of this as RFC 265, restated FTP for additional review, while RFC 281 suggested further changes. The use of a "Set Data Type" transaction was proposed in RFC 294 in January 1982.

RFC 354 obsoleted RFCs 264 and 265. The File Transfer Protocol was now defined as a protocol for file transfer between HOSTs on the ARPANET, with the primary function of FTP defined as transfering files efficiently and reliably among hosts and allowing the convenient use of remote file storage capabilities. RFC 385 further commented on errors, emphasis points, and additions to the protocol, while RFC 414 provided a status report on the working server and user FTPs. RFC 430, issued in 1973, (among other RFCs too numerous to mention) presented further comments on FTP. Finally, an "official" FTP document was published as RFC 454.

By July 1973, considerable changes from the last versions of FTP were made, but the general structure remained the same. RFC 542 was published as a new "official" specification to reflect these changes. However, many implementations based on the older specification were not updated.

In 1974, RFCs 607 and 614 continued comments on FTP. RFC 624 proposed further design changes and minor modifications. In 1975, RFC 686 entitled, "Leaving Well Enough Alone", discussed the differences between all of the early and later versions of FTP. RFC 691 presented a minor revision of RFC 686, regarding the subject of print files.

Motivated by the transition from the NCP to the TCP as the underlying protocol, a phoenix was born out of all of the above efforts in RFC 765 as the specification of FTP for use on TCP.

This current edition of the FTP specification is intended to correct some minor documentation errors, to improve the explanation of some protocol features, and

to add some new optional commands.

In particular, the following new optional commands are included in this edition of the specification:

CDUP - Change to Parent Directory
SMNT - Structure Mount
STOU - Store Unique
RMD - Remove Directory
MKD - Make Directory
PWD - Print Directory
SYST - System

This specification is compatible with the previous edition. A program implemented in conformance to the previous specification should automatically be in conformance to this specification.

2.2 TERMINOLOGY

ASCII

The ASCII character set is as defined in the ARPA-Internet Protocol Handbook. In FTP, ASCII characters are defined to be the lower half of an eight-bit code set (i.e., the most significant bit is zero).

access controls

Access controls define users' access privileges to the use of a system, and to the files in that system. Access controls are necessary to prevent unauthorized or accidental use of files. It is the prerogative of a server-FTP process to invoke access controls.

byte size

There are two byte sizes of interest in FTP: the logical byte size of the file, and the transfer byte size used for the transmission of the data. The transfer byte size is always 8 bits. The transfer byte size is not necessarily the byte size in which data is to be stored in a system, nor the logical byte size for interpretation of the structure of the data.

control connection

The communication path between the USER-PI and SERVER-PI for the exchange of commands and replies. This connection follows the Telnet Protocol.

data connection

A full duplex connection over which data is transferred, in a specified mode and type. The data transferred may be a part of a file, an entire file or a number of files. The path may be between a server-DTP and a user-DTP, or between two server-DTPs.

data port

The passive data transfer process "listens" on the data port for a connection from the active transfer process in order to open the data connection.

DTP

The data transfer process establishes and manages the data connection. The DTP can be passive or active.

End-of-Line

The end-of-line sequence defines the separation of printing lines. The sequence is Carriage Return, followed by Line Feed.

EOF

The end-of-file condition that defines the end of a file being transferred.

EOR

The end-of-record condition that defines the end of a record being transferred.

error recovery

A procedure that allows a user to recover from certain errors such as failure of either host system or transfer process. In FTP, error recovery may involve restarting a file transfer at a given checkpoint.

FTP commands

A set of commands that comprise the control information flowing from the user-FTP to the server-FTP process.

file

An ordered set of computer data (including programs), of arbitrary length, uniquely identified by a pathname.

mode

The mode in which data is to be transferred via the data connection. The mode defines the data format during transfer including EOR and EOF. The transfer modes defined in FTP are described in the Section on Transmission Modes.

NVT

The Network Virtual Terminal as defined in the Telnet Protocol.

NVFS

The Network Virtual File System. A concept which defines a standard network file system with standard commands and pathname conventions.

page

A file may be structured as a set of independent parts called pages. FTP supports the transmission of discontinuous files as independent indexed pages.

pathname

Pathname is defined to be the character string which must be input to a file system by a user in order to identify a file. Pathname normally contains device and/or directory names, and file name specification. FTP does not yet specify a standard pathname convention. Each user must follow the file naming conventions of the file systems involved in the transfer.

PI

The protocol interpreter. The user and server sides of the protocol have distinct roles implemented in a user-PI and a server-PI.

record

A sequential file may be structured as a number of contiguous parts called records. Record structures are supported by FTP but a file need not have record structure.

reply

A reply is an acknowledgment (positive or negative) sent from server to user via the control connection in response to FTP commands. The general form of a reply is a completion code (including error codes) followed by a text string. The codes are for use by programs and the text is usually intended for human users.

server-DTP

The data transfer process, in its normal "active" state, establishes the data connection with the "listening" data port. It sets up parameters for transfer and storage, and transfers data on command from its PI. The DTP can be placed in a "passive" state to listen for, rather than initiate a connection on the data port.

server-FTP process

A process or set of processes which perform the function of file transfer in cooperation with a user-FTP process and, possibly, another server. The functions consist of a protocol interpreter (PI) and a data transfer process (DTP).

server-PI

The server protocol interpreter "listens" on Port L for a connection from a user-PI and establishes a control communication connection. It receives standard FTP commands from the user-PI, sends replies, and governs the server-DTP.

type

The data representation type used for data transfer and storage. Type implies certain transformations between the time of data storage and data transfer. The representation types defined in FTP are described in the Section on Establishing Data Connections.

user

A person or a process on behalf of a person wishing to obtain file transfer service. The human user may interact directly with a server-FTP process, but use of a user-FTP process is preferred since the protocol design is weighted towards automata.

user-DTP

The data transfer process "listens" on the data port for a connection from a server-FTP process. If two servers are transferring data between them, the user-DTP is inactive.

user-FTP process

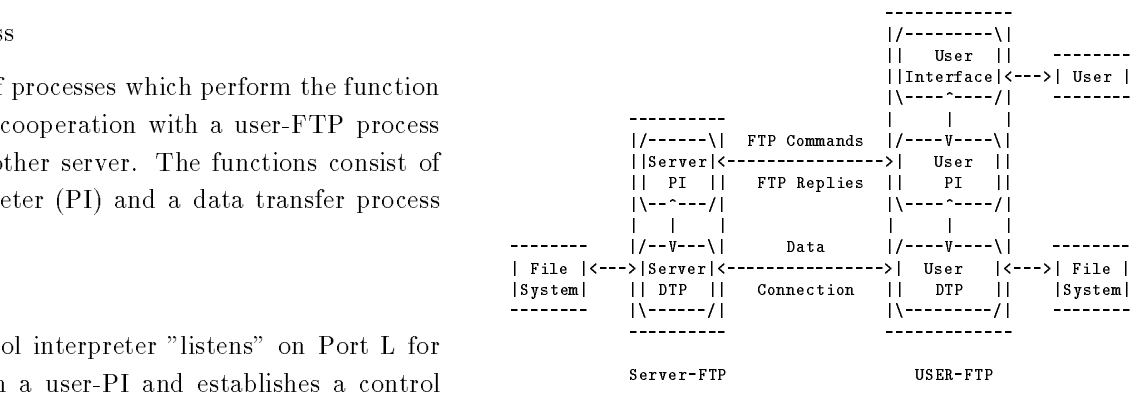
A set of functions including a protocol interpreter, a data transfer process and a user interface which together perform the function of file transfer in cooperation with one or more server-FTP processes. The user interface allows a local language to be used in the command-reply dialogue with the user.

user-PI

The user protocol interpreter initiates the control connection from its port U to the server-FTP process, initiates FTP commands, and governs the user-DTP if that process is part of the file transfer.

2.3 THE FTP MODEL

With the above definitions in mind, the following model (shown in Figure 1) may be diagrammed for an FTP service.



NOTES: 1. The data connection may be used in either direction.
2. The data connection need not exist all of the time.

Figure 1 Model for FTP Use

In the model described in Figure 1, the user-protocol interpreter initiates the control connection. The control connection follows the Telnet protocol. At the initiation

of the user, standard FTP commands are generated by the user-PI and transmitted to the server process via the control connection. (The user may establish a direct control connection to the server-FTP, from a TAC terminal for example, and generate standard FTP commands independently, bypassing the user-FTP process.) Standard replies are sent from the server-PI to the user-PI over the control connection in response to the commands.

The FTP commands specify the parameters for the data connection (data port, transfer mode, representation type, and structure) and the nature of file system operation (store, retrieve, append, delete, etc.). The user-DTP or its designate should "listen" on the specified data port, and the server initiate the data connection and data transfer in accordance with the specified parameters. It should be noted that the data port need not be in the same host that initiates the FTP commands via the control connection, but the user or the user-FTP process must ensure a "listen" on the specified data port. It ought to also be noted that the data connection may be used for simultaneous sending and receiving.

In another situation a user might wish to transfer files between two hosts, neither of which is a local host. The user sets up control connections to the two servers and then arranges for a data connection between them. In this manner, control information is passed to the user-PI but data is transferred between the server data transfer processes. Following is a model of this server-server interaction.

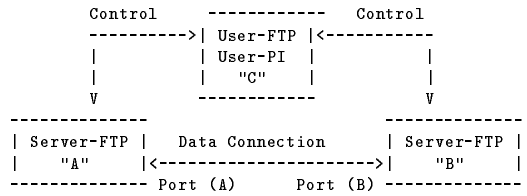


Figure 2

The protocol requires that the control connections be open while data transfer is in progress. It is the responsibility of the user to request the closing of the control connections when finished using the FTP service, while it is the server who takes the action. The server may abort data transfer if the control connections are closed without command.

The Relationship between FTP and Telnet:

> The FTP uses the Telnet protocol on the control connection. This can be achieved in two ways: first, the user-PI or the server-PI may implement the rules of the Telnet Protocol directly in their own procedures; or, second, the user-PI or the server-PI may make use of the existing Telnet module in the system.
>
> Ease of implementaion, sharing code, and modular programming argue for the second approach. Efficiency and independence argue for the first approach. In practice, FTP relies on very little of the Telnet Protocol, so the first approach does not necessarily involve a large amount of code.

3 DATA TRANSFER FUNCTIONS

Files are transferred only via the data connection. The control connection is used for the transfer of commands, which describe the functions to be performed, and the replies to these commands (see the Section on FTP Replies). Several commands are concerned with the transfer of data between hosts. These data transfer commands include the MODE command which specify how the bits of the data are to be transmitted, and the STRUcture and TYPE commands, which are used to define the way in which the data are to be represented. The transmission and representation are basically independent but the "Stream" transmission mode is dependent on the file structure attribute and if "Compressed" transmission mode is used, the nature of the filler byte depends on the representation type.

3.1 DATA REPRESENTATION AND STORAGE

Data is transferred from a storage device in the sending host to a storage device in the receiving host. Often it is necessary to perform certain transformations on the data because data storage representations in the two systems are different. For example, NVT-ASCII has different data storage representations in different systems. DEC TOPS-20s's generally store NVT-ASCII as five 7-bit ASCII characters, left-justified in a 36-bit word. IBM Mainframe's store NVT-ASCII as 8-bit EBCDIC codes. Multics stores NVT-ASCII as four 9-bit characters in a 36-bit word. It is desirable to convert characters into the standard NVT-ASCII representation when transmitting text between dissimilar systems. The sending and receiving sites would have to perform the necessary transformations between the standard representation and their internal representations.

A different problem in representation arises when transmitting binary data (not character codes) between host systems with different word lengths. It is not always clear how the sender should send data, and the receiver store it. For example, when transmitting 32-bit bytes from a 32-bit word-length system to a 36-bit word-length system, it may be desirable (for reasons of efficiency and usefulness) to store the 32-bit bytes right-justified in a 36-bit word in the latter system. In any case, the user

should have the option of specifying data representation and transformation functions. It should be noted that FTP provides for very limited data type representations. Transformations desired beyond this limited capability should be performed by the user directly.

3.1.1. DATA TYPES

Data representations are handled in FTP by a user specifying a representation type. This type may implicitly (as in ASCII or EBCDIC) or explicitly (as in Local byte) define a byte size for interpretation which is referred to as the "logical byte size." Note that this has nothing to do with the byte size used for transmission over the data connection, called the "transfer byte size", and the two should not be confused. For example, NVT-ASCII has a logical byte size of 8 bits. If the type is Local byte, then the TYPE command has an obligatory second parameter specifying the logical byte size. The transfer byte size is always 8 bits.

3.1.1.1. ASCII TYPE

This is the default type and must be accepted by all FTP implementations. It is intended primarily for the transfer of text files, except when both hosts would find the EBCDIC type more convenient.

The sender converts the data from an internal character representation to the standard 8-bit NVT-ASCII representation (see the Telnet specification). The receiver will convert the data from the standard form to his own internal form.

In accordance with the NVT standard, the <CRLF> sequence should be used where necessary to denote the end of a line of text. (See the discussion of file structure at the end of the Section on Data Representation and Storage.)

Using the standard NVT-ASCII representation means that data must be interpreted as 8-bit bytes.

The Format parameter for ASCII and EBCDIC types is discussed below.

3.1.1.2. EBCDIC TYPE

This type is intended for efficient transfer between hosts which use EBCDIC for their internal character representation.

For transmission, the data are represented as 8-bit EBCDIC characters. The character code is the only difference between the functional specifications of EBCDIC and ASCII types.

End-of-line (as opposed to end-of-record–see the discussion of structure) will probably be rarely used with EBCDIC type for purposes of denoting structure, but where it is necessary the <NL> character should be used.

3.1.1.3. IMAGE TYPE

The data are sent as contiguous bits which, for transfer, are packed into the 8-bit transfer bytes. The receiving site must store the data as contiguous bits. The structure of the storage system might necessitate the padding of the file (or of each record, for a record-structured file) to some convenient boundary (byte, word or block). This padding, which must be all zeros, may occur only at the end of the file (or at the end of each record) and there must be a way of identifying the padding bits so that they may be stripped off if the file is retrieved. The padding transformation should be well publicized to enable a user to process a file at the storage site.

Image type is intended for the efficient storage and retrieval of files and for the transfer of binary data. It is recommended that this type be accepted by all FTP implementations.

3.1.1.4. LOCAL TYPE

The data is transferred in logical bytes of the size specified by the obligatory second parameter, Byte size. The value of Byte size must be a decimal integer; there is no default value. The logical byte size is not necessarily the same as the transfer byte size. If there is a difference in byte sizes, then the logical bytes should be packed contiguously, disregarding transfer byte boundaries and with any necessary padding at the end.

When the data reaches the receiving host, it will be transformed in a manner dependent on the logical byte size and the particular host. This transformation must be invertible (i.e., an identical file can be retrieved if the same parameters are used) and should be well publicized by the FTP implementors.

For example, a user sending 36-bit floating-point numbers to a host with a 32-bit word could send that data as Local byte with a logical byte size of 36. The receiving host would then be expected to store the logical bytes so that they could be easily manipulated; in this example putting the 36-bit logical bytes into 64-bit double words should suffice.

In another example, a pair of hosts with a 36-bit word size may send data to one another in words by using TYPE L 36. The data would be sent in the 8-bit transmission bytes packed so that 9 transmission bytes carried two host words.

3.1.1.5. FORMAT CONTROL

The types ASCII and EBCDIC also take a second (optional) parameter; this is to indicate what kind of vertical format control, if any, is associated with a file. The following data representation types are defined in FTP:

A character file may be transferred to a host for one of three purposes: for printing, for storage and later retrieval, or for processing. If a file is sent for printing, the receiving host must know how the vertical format

control is represented. In the second case, it must be possible to store a file at a host and then retrieve it later in exactly the same form. Finally, it should be possible to move a file from one host to another and process the file at the second host without undue trouble. A single ASCII or EBCDIC format does not satisfy all these conditions. Therefore, these types have a second parameter specifying one of the following three formats:

3.1.1.5.1. NON PRINT

This is the default format to be used if the second (format) parameter is omitted. Non-print format must be accepted by all FTP implementations.

The file need contain no vertical format information. If it is passed to a printer process, this process may assume standard values for spacing and margins.

Normally, this format will be used with files destined for processing or just storage.

3.1.1.5.2. TELNET FORMAT CONTROLS

The file contains ASCII/EBCDIC vertical format controls (i.e., <CR>, <LF>, <NL>, <VT>, <FF>) which the printer process will interpret appropriately. <CRLF>, in exactly this sequence, also denotes end-of-line.

3.1.1.5.2. CARRIAGE CONTROL (ASA)

The file contains ASA (FORTRAN) vertical format control characters. (See RFC 740 Appendix C; and Communications of the ACM, Vol. 7, No. 10, p. 606, October 1964.) In a line or a record formatted according to the ASA Standard, the first character is not to be printed. Instead, it should be used to determine the vertical movement of the paper which should take place before the rest of the record is printed.

The ASA Standard specifies the following control characters:

Character	Vertical Spacing
blank	Move paper up one line
0	Move paper up two lines
1	Move paper to top of next page
+	No movement, i.e., overprint

Clearly there must be some way for a printer process to distinguish the end of the structural entity. If a file has record structure (see below) this is no problem; records will be explicitly marked during transfer and storage. If the file has no record structure, the <CRLF> end-of-line sequence is used to separate printing lines, but these format effectors are overridden by the ASA controls.

3.1.2. DATA STRUCTURES

In addition to different representation types, FTP allows the structure of a file to be specified. Three file structures are defined in FTP:

file-structure, where there is no internal structure and the file is considered to be a continuous sequence of data bytes,

record-structure, where the file is made up of sequential records,

and page-structure, where the file is made up of independent indexed pages.

File-structure is the default to be assumed if the STRUcture command has not been used but both file and record structures must be accepted for "text" files (i.e., files with TYPE ASCII or EBCDIC) by all FTP implementations. The structure of a file will affect both the transfer mode of a file (see the Section on Transmission Modes) and the interpretation and storage of the file.

The "natural" structure of a file will depend on which host stores the file. A source-code file will usually be stored on an IBM Mainframe in fixed length records but on a DEC TOPS-20 as a stream of characters partitioned into lines, for example by <CRLF>. If the transfer of files between such disparate sites is to be useful, there must be some way for one site to recognize the other's assumptions about the file.

With some sites being naturally file-oriented and others naturally record-oriented there may be problems if a file with one structure is sent to a host oriented to the other. If a text file is sent with record-structure to a host which is file oriented, then that host should apply an internal transformation to the file based on the record structure. Obviously, this transformation should be useful, but it must also be invertible so that an identical file may be retrieved using record structure.

In the case of a file being sent with file-structure to a record-oriented host, there exists the question of what criteria the host should use to divide the file into records which can be processed locally. If this division is necessary, the FTP implementation should use the end-of-line sequence, <CRLF> for ASCII, or <NL> for EBCDIC text files, as the delimiter. If an FTP implementation adopts this technique, it must be prepared to reverse the transformation if the file is retrieved with file-structure.

3.1.2.1. FILE STRUCTURE

File structure is the default to be assumed if the STRUcture command has not been used.

In file-structure there is no internal structure and the file is considered to be a continuous sequence of data bytes.

3.1.2.2. RECORD STRUCTURE

Record structures must be accepted for "text" files (i.e., files with TYPE ASCII or EBCDIC) by all FTP implementations.

In record-structure the file is made up of sequential records.

3.1.2.3. PAGE STRUCTURE

To transmit files that are discontinuous, FTP defines a page structure. Files of this type are sometimes known as "random access files" or even as "holey files". In these files there is sometimes other information associated with the file as a whole (e.g., a file descriptor), or with a section of the file (e.g., page access controls), or both. In FTP, the sections of the file are called pages.

To provide for various page sizes and associated information, each page is sent with a page header. The page header has the following defined fields:

Header Length

The number of logical bytes in the page header including this byte. The minimum header length is 4.

Page Index

The logical page number of this section of the file. This is not the transmission sequence number of this page, but the index used to identify this page of the file.

Data Length

The number of logical bytes in the page data. The minimum data length is 0.

Page Type

The type of page this is. The following page types are defined:

0 = Last Page

This is used to indicate the end of a paged structured transmission. The header length must be 4, and the data length must be 0.

1 = Simple Page

This is the normal type for simple paged files with no page level associated control information. The header length must be 4.

2 = Descriptor Page

This type is used to transmit the descriptive information for the file as a whole.

3 = Access Controlled Page

This type includes an additional header field for paged files with page level access control information. The header length must be 5.

Optional Fields

Further header fields may be used to supply per page control information, for example, per page access control.

All fields are one logical byte in length. The logical byte size is specified by the TYPE command. See Appendix I for further details and a specific case at the page structure.

A note of caution about parameters: a file must be stored and retrieved with the same parameters if the retrieved version is to be identical to the version originally transmitted. Conversely, FTP implementations must return a file identical to the original if the parameters used to store and retrieve a file are the same.

3.2 ESTABLISHING DATA CONNECTIONS

The mechanics of transferring data consists of setting up the data connection to the appropriate ports and choosing the parameters for transfer. Both the user and the server-DTPs have a default data port. The user-process default data port is the same as the control connection port (i.e., U). The server-process default data port is the port adjacent to the control connection port (i.e., L-1).

The transfer byte size is 8-bit bytes. This byte size is relevant only for the actual transfer of the data; it has no bearing on representation of the data within a host's file system.

The passive data transfer process (this may be a user-DTP or a second server-DTP) shall "listen" on the data port prior to sending a transfer request command. The FTP request command determines the direction of the data transfer. The server, upon receiving the transfer request, will initiate the data connection to the port. When the connection is established, the data transfer begins between DTP's, and the server-PI sends a confirming reply to the user-PI.

Every FTP implementation must support the use of the default data ports, and only the USER-PI can initiate a change to non-default ports.

It is possible for the user to specify an alternate data port by use of the PORT command. The user may want a file dumped on a TAC line printer or retrieved from a third party host. In the latter case, the user-PI sets up control connections with both server-PI's. One server is then told (by an FTP command) to "listen" for a connection which the other will initiate. The user-PI sends one server-PI a PORT command indicating the data port of the other. Finally, both are sent the appropriate transfer commands. The exact sequence of commands and

replies sent between the user-controller and the servers is defined in the Section on FTP Replies.

In general, it is the server's responsibility to maintain the data connection–to initiate it and to close it. The exception to this is when the user-DTP is sending the data in a transfer mode that requires the connection to be closed to indicate EOF. The server MUST close the data connection under the following conditions:

1. The server has completed sending data in a transfer mode that requires a close to indicate EOF.
2. The server receives an ABORT command from the user.
3. The port specification is changed by a command from the user.
4. The control connection is closed legally or otherwise.
5. An irrecoverable error condition occurs.

Otherwise the close is a server option, the exercise of which the server must indicate to the user-process by either a 250 or 226 reply only.

3.3 DATA CONNECTION MANAGEMENT

Default Data Connection Ports: All FTP implementations must support use of the default data connection ports, and only the User-PI may initiate the use of non-default ports.

Negotiating Non-Default Data Ports: The User-PI may specify a non-default user side data port with the PORT command. The User-PI may request the server side to identify a non-default server side data port with the PASV command. Since a connection is defined by the pair of addresses, either of these actions is enough to get a different data connection, still it is permitted to do both commands to use new ports on both ends of the data connection.

Reuse of the Data Connection: When using the stream mode of data transfer the end of the file must be indicated by closing the connection. This causes a problem if multiple files are to be transfered in the session, due to need for TCP to hold the connection record for a time out period to guarantee the reliable communication. Thus the connection can not be reopened at once.

There are two solutions to this problem. The first is to negotiate a non-default port. The second is to use another transfer mode.

A comment on transfer modes. The stream transsfer mode is inherently unreliable, since one can not determine if the connection closed prematurely or not. The other transfer modes (Block, Compressed) do not close the connection to indicate the end of file. They have enough FTP encoding that the data connection can be parsed to determine the end of the file. Thus using these modes one can leave the data connection open for multiple file transfers.

3.4 TRANSMISSION MODES

The next consideration in transferring data is choosing the appropriate transmission mode. There are three modes: one which formats the data and allows for restart procedures; one which also compresses the data for efficient transfer; and one which passes the data with little or no processing. In this last case the mode interacts with the structure attribute to determine the type of processing. In the compressed mode, the representation type determines the filler byte.

All data transfers must be completed with an end-of-file (EOF) which may be explicitly stated or implied by the closing of the data connection. For files with record structure, all the end-of-record markers (EOR) are explicit, including the final one. For files transmitted in page structure a "last-page" page type is used.

NOTE: In the rest of this section, byte means "transfer byte" except where explicitly stated otherwise.

For the purpose of standardized transfer, the sending host will translate its internal end of line or end of record denotation into the representation prescribed by the transfer mode and file structure, and the receiving host will perform the inverse translation to its internal denotation. An IBM Mainframe record count field may not be recognized at another host, so the end-of-record information may be transferred as a two byte control code in Stream mode or as a flagged bit in a Block or Compressed mode descriptor. End-of-line in an ASCII or EBCDIC file with no record structure should be indicated by `<CRLF>` or `<NL>`, respectively. Since these transformations imply extra work for some systems, identical systems transferring non-record structured text files might wish to use a binary representation and stream mode for the transfer.

The following transmission modes are defined in FTP:

3.4.1. STREAM MODE

The data is transmitted as a stream of bytes. There is no restriction on the representation type used; record structures are allowed.

In a record structured file EOR and EOF will each be indicated by a two-byte control code. The first byte of the control code will be all ones, the escape character. The second byte will have the low order bit on and zeros elsewhere for EOR and the second low order bit on for EOF; that is, the byte will have value 1 for EOR and value 2 for EOF. EOR and EOF may be indicated together on the last byte transmitted by turning both low order bits on (i.e., the value 3). If a byte of all ones was

intended to be sent as data, it should be repeated in the second byte of the control code.

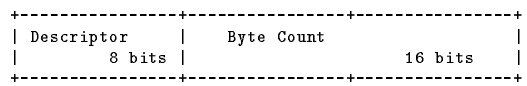
If the structure is a file structure, the EOF is indicated by the sending host closing the data connection and all bytes are data bytes.

3.4.2. BLOCK MODE

The file is transmitted as a series of data blocks preceded by one or more header bytes. The header bytes contain a count field, and descriptor code. The count field indicates the total length of the data block in bytes, thus marking the beginning of the next data block (there are no filler bits). The descriptor code defines: last block in the file (EOF) last block in the record (EOR), restart marker (see the Section on Error Recovery and Restart) or suspect data (i.e., the data being transferred is suspected of errors and is not reliable). This last code is NOT intended for error control within FTP. It is motivated by the desire of sites exchanging certain types of data (e.g., seismic or weather data) to send and receive all the data despite local errors (such as "magnetic tape read errors"), but to indicate in the transmission that certain portions are suspect). Record structures are allowed in this mode, and any representation type may be used.

The header consists of the three bytes. Of the 24 bits of header information, the 16 low order bits shall represent byte count, and the 8 high order bits shall represent descriptor codes as shown below.

Block Header



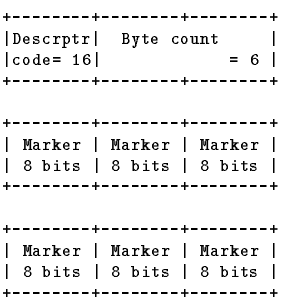
The descriptor codes are indicated by bit flags in the descriptor byte. Four codes have been assigned, where each code number is the decimal value of the corresponding bit in the byte.

Code	Meaning
128	End of data block is EOR
64	End of data block is EOF
32	Suspected errors in data block
16	Data block is a restart marker

With this encoding, more than one descriptor coded condition may exist for a particular block. As many bits as necessary may be flagged.

The restart marker is embedded in the data stream as an integral number of 8-bit bytes representing printable characters in the language being used over the control connection (e.g., default–NVT-ASCII). <SP> (Space, in the appropriate language) must not be used WITHIN a restart marker.

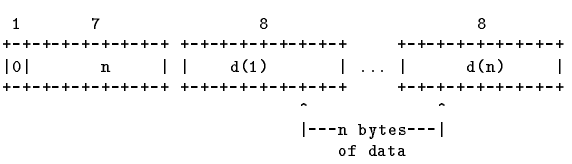
For example, to transmit a six-character marker, the following would be sent:



3.4.3. COMPRESSED MODE

There are three kinds of information to be sent: regular data, sent in a byte string; compressed data, consisting of replications or filler; and control information, sent in a two-byte escape sequence. If n>0 bytes (up to 127) of regular data are sent, these n bytes are preceded by a byte with the left-most bit set to 0 and the right-most 7 bits containing the number n.

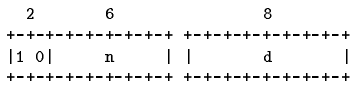
Byte string:



String of n data bytes d(1),..., d(n) Count n must be positive.

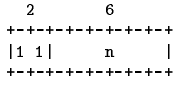
To compress a string of n replications of the data byte d, the following 2 bytes are sent:

Replicated Byte:



A string of n filler bytes can be compressed into a single byte, where the filler byte varies with the representation type. If the type is ASCII or EBCDIC the filler byte is ¡SP¿ (Space, ASCII code 32, EBCDIC code 64). If the type is Image or Local byte the filler is a zero byte.

Filler String:



The escape sequence is a double byte, the first of which is the escape byte (all zeros) and the second of which contains descriptor codes as defined in Block mode. The descriptor codes have the same meaning as in Block mode and apply to the succeeding string of bytes.

Compressed mode is useful for obtaining increased bandwidth on very large network transmissions at a little extra CPU cost. It can be most effectively used to reduce the size of printer files such as those generated by RJE hosts.

3.5 ERROR RECOVERY AND RESTART

There is no provision for detecting bits lost or scrambled in data transfer; this level of error control is handled by the TCP. However, a restart procedure is provided to protect users from gross system failures (including failures of a host, an FTP-process, or the underlying network).

The restart procedure is defined only for the block and compressed modes of data transfer. It requires the sender of data to insert a special marker code in the data stream with some marker information. The marker information has meaning only to the sender, but must consist of printable characters in the default or negotiated language of the control connection (ASCII or EBCDIC). The marker could represent a bit-count, a record-count, or any other information by which a system may identify a data checkpoint. The receiver of data, if it implements the restart procedure, would then mark the corresponding position of this marker in the receiving system, and return this information to the user.

In the event of a system failure, the user can restart the data transfer by identifying the marker point with the FTP restart procedure. The following example illustrates the use of the restart procedure.

The sender of the data inserts an appropriate marker block in the data stream at a convenient point. The receiving host marks the corresponding data point in its file system and conveys the last known sender and receiver marker information to the user, either directly or over the control connection in a 110 reply (depending on who is the sender). In the event of a system failure, the user or controller process restarts the server at the last server marker by sending a restart command with server's marker code as its argument. The restart command is transmitted over the control connection and is immediately followed by the command (such as RETR, STOR or LIST) which was being executed when the system failure occurred.