Network Working Group J. Postel

Request for Comments: 959 J. Reynolds

ISI

Obsoletes RFC: 765 (IEN 149) October 1985

FILE TRANSFER PROTOCOL (FTP)

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.

Postel & Reynolds [Page 1]

RFC 959 October 1985

File Transfer Protocol

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.

Postel & Reynolds [Page 2]

RFC 959 October 1985

File Transfer Protocol

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.

Postel & Reynolds [Page 3]

RFC 959 October 1985

File Transfer Protocol

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.

Postel & Reynolds [Page 4]

RFC 959 October 1985

File Transfer Protocol

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.

Postel & Reynolds [Page 5]

RFC 959 October 1985

File Transfer Protocol

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.

Postel & Reynolds [Page 6]

RFC 959 October 1985

File Transfer Protocol

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.

Postel & Reynolds [Page 7]

RFC 959 October 1985

File Transfer Protocol

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.

-------------

|/---------\|

|| User || --------

||Interface|<--->| User |

|\----^----/| --------

---------- | | |

|/------\| FTP Commands |/----V----\|

||Server|<---------------->| User ||

|| PI || FTP Replies || PI ||

|\--^---/| |\----^----/|

| | | | | |

-------- |/--V---\| Data |/----V----\| --------

| File |<--->|Server|<---------------->| User |<--->| File |

|System| || DTP || Connection || DTP || |System|

-------- |\------/| |\---------/| --------

---------- -------------

Server-FTP USER-FTP

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

Postel & Reynolds [Page 8]

RFC 959 October 1985

File Transfer Protocol

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.

Control ------------ Control

---------->| User-FTP |<-----------

| | User-PI | |

| | "C" | |

V ------------ V

-------------- --------------

| Server-FTP | Data Connection | Server-FTP |

| "A" |<---------------------->| "B" |

-------------- Port (A) Port (B) --------------

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

Postel & Reynolds [Page 9]

RFC 959 October 1985

File Transfer Protocol

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

Postel & Reynolds [Page 10]

RFC 959 October 1985

File Transfer Protocol

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.

Postel & Reynolds [Page 11]

RFC 959 October 1985

File Transfer Protocol

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.

Postel & Reynolds [Page 12]

RFC 959 October 1985

File Transfer Protocol

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.

Postel & Reynolds [Page 13]

RFC 959 October 1985

File Transfer Protocol

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.

Postel & Reynolds [Page 14]

RFC 959 October 1985

File Transfer Protocol

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,

Postel & Reynolds [Page 15]

RFC 959 October 1985

File Transfer Protocol

<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.

Postel & Reynolds [Page 16]

RFC 959 October 1985

File Transfer Protocol

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

Postel & Reynolds [Page 17]

RFC 959 October 1985

File Transfer Protocol

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

Postel & Reynolds [Page 18]

RFC 959 October 1985

File Transfer Protocol

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 transfer mode is

Postel & Reynolds [Page 19]

RFC 959 October 1985

File Transfer Protocol

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.

Postel & Reynolds [Page 20]

RFC 959 October 1985

File Transfer Protocol

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.

Postel & Reynolds [Page 21]

RFC 959 October 1985

File Transfer Protocol

Block Header

+----------------+----------------+----------------+

| Descriptor | Byte Count |

| 8 bits | 16 bits |

+----------------+----------------+----------------+

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:

+--------+--------+--------+

|Descrptr| Byte count |

|code= 16| = 6 |

+--------+--------+--------+

+--------+--------+--------+

| Marker | Marker | Marker |

| 8 bits | 8 bits | 8 bits |

+--------+--------+--------+

+--------+--------+--------+

| Marker | Marker | Marker |

| 8 bits | 8 bits | 8 bits |

+--------+--------+--------+

Postel & Reynolds [Page 22]

RFC 959 October 1985

File Transfer Protocol

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:

1 7 8 8

+-+-+-+-+-+-+-+-+ +-+-+-+-+-+-+-+-+ +-+-+-+-+-+-+-+-+

|0| n | | d(1) | ... | d(n) |

+-+-+-+-+-+-+-+-+ +-+-+-+-+-+-+-+-+ +-+-+-+-+-+-+-+-+

^ ^

|---n bytes---|

of data

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:

2 6 8

+-+-+-+-+-+-+-+-+ +-+-+-+-+-+-+-+-+

|1 0| n | | d |

+-+-+-+-+-+-+-+-+ +-+-+-+-+-+-+-+-+

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:

2 6

+-+-+-+-+-+-+-+-+

|1 1| n |

+-+-+-+-+-+-+-+-+

The escape sequence is a double byte, the first of which is the

Postel & Reynolds [Page 23]

RFC 959 October 1985

File Transfer Protocol

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

Postel & Reynolds [Page 24]

RFC 959 October 1985

File Transfer Protocol

connection and is immediately followed by the command (such as

RETR, STOR or LIST) which was being executed when the system

failure occurred.

4. FILE TRANSFER FUNCTIONS

The communication channel from the user-PI to the server-PI is

established as a TCP connection from the user to the standard server

port. The user protocol interpreter is responsible for sending FTP

commands and interpreting the replies received; the server-PI

interprets commands, sends replies and directs its DTP to set up the

data connection and transfer the data. If the second party to the

data transfer (the passive transfer process) is the user-DTP, then it

is governed through the internal protocol of the user-FTP host; if it

is a second server-DTP, then it is governed by its PI on command from

the user-PI. The FTP replies are discussed in the next section. In

the description of a few of the commands in this section, it is

helpful to be explicit about the possible replies.

4.1. FTP COMMANDS

4.1.1. ACCESS CONTROL COMMANDS

The following commands specify access control identifiers

(command codes are shown in parentheses).

USER NAME (USER)

The argument field is a Telnet string identifying the user.

The user identification is that which is required by the

server for access to its file system. This command will

normally be the first command transmitted by the user after

the control connections are made (some servers may require

this). Additional identification information in the form of

a password and/or an account command may also be required by

some servers. Servers may allow a new USER command to be

entered at any point in order to change the access control

and/or accounting information. This has the effect of

flushing any user, password, and account information already

supplied and beginning the login sequence again. All

transfer parameters are unchanged and any file transfer in

progress is completed under the old access control

parameters.

Postel & Reynolds [Page 25]

RFC 959 October 1985

File Transfer Protocol

PASSWORD (PASS)

The argument field is a Telnet string specifying the user's

password. This command must be immediately preceded by the

user name command, and, for some sites, completes the user's

identification for access control. Since password

information is quite sensitive, it is desirable in general

to "mask" it or suppress typeout. It appears that the

server has no foolproof way to achieve this. It is

therefore the responsibility of the user-FTP process to hide

the sensitive password information.

ACCOUNT (ACCT)

The argument field is a Telnet string identifying the user's

account. The command is not necessarily related to the USER

command, as some sites may require an account for login and

others only for specific access, such as storing files. In

the latter case the command may arrive at any time.

There are reply codes to differentiate these cases for the

automation: when account information is required for login,

the response to a successful PASSword command is reply code

332. On the other hand, if account information is NOT

required for login, the reply to a successful PASSword

command is 230; and if the account information is needed for

a command issued later in the dialogue, the server should

return a 332 or 532 reply depending on whether it stores

(pending receipt of the ACCounT command) or discards the

command, respectively.

CHANGE WORKING DIRECTORY (CWD)

This command allows the user to work with a different

directory or dataset for file storage or retrieval without

altering his login or accounting information. Transfer

parameters are similarly unchanged. The argument is a

pathname specifying a directory or other system dependent

file group designator.

CHANGE TO PARENT DIRECTORY (CDUP)

This command is a special case of CWD, and is included to

simplify the implementation of programs for transferring

directory trees between operating systems having different

Postel & Reynolds [Page 26]

RFC 959 October 1985

File Transfer Protocol

syntaxes for naming the parent directory. The reply codes

shall be identical to the reply codes of CWD. See

Appendix II for further details.

STRUCTURE MOUNT (SMNT)

This command allows the user to mount a different file

system data structure without altering his login or

accounting information. Transfer parameters are similarly

unchanged. The argument is a pathname specifying a

directory or other system dependent file group designator.

REINITIALIZE (REIN)

This command terminates a USER, flushing all I/O and account

information, except to allow any transfer in progress to be

completed. All parameters are reset to the default settings

and the control connection is left open. This is identical

to the state in which a user finds himself immediately after

the control connection is opened. A USER command may be

expected to follow.

LOGOUT (QUIT)

This command terminates a USER and if file transfer is not

in progress, the server closes the control connection. If

file transfer is in progress, the connection will remain

open for result response and the server will then close it.

If the user-process is transferring files for several USERs

but does not wish to close and then reopen connections for

each, then the REIN command should be used instead of QUIT.

An unexpected close on the control connection will cause the

server to take the effective action of an abort (ABOR) and a

logout (QUIT).

4.1.2. TRANSFER PARAMETER COMMANDS

All data transfer parameters have default values, and the

commands specifying data transfer parameters are required only

if the default parameter values are to be changed. The default

value is the last specified value, or if no value has been

specified, the standard default value is as stated here. This

implies that the server must "remember" the applicable default

values. The commands may be in any order except that they must

precede the FTP service request. The following commands

specify data transfer parameters:

Postel & Reynolds [Page 27]

RFC 959 October 1985

File Transfer Protocol

DATA PORT (PORT)

The argument is a HOST-PORT specification for the data port

to be used in data connection. There are defaults for both

the user and server data ports, and under normal

circumstances this command and its reply are not needed. If

this command is used, the argument is the concatenation of a

32-bit internet host address and a 16-bit TCP port address.

This address information is broken into 8-bit fields and the

value of each field is transmitted as a decimal number (in

character string representation). The fields are separated

by commas. A port command would be:

PORT h1,h2,h3,h4,p1,p2

where h1 is the high order 8 bits of the internet host

address.

PASSIVE (PASV)

This command requests the server-DTP to "listen" on a data

port (which is not its default data port) and to wait for a

connection rather than initiate one upon receipt of a

transfer command. The response to this command includes the

host and port address this server is listening on.

REPRESENTATION TYPE (TYPE)

The argument specifies the representation type as described

in the Section on Data Representation and Storage. Several

types take a second parameter. The first parameter is

denoted by a single Telnet character, as is the second

Format parameter for ASCII and EBCDIC; the second parameter

for local byte is a decimal integer to indicate Bytesize.

The parameters are separated by a <SP> (Space, ASCII code

32).

The following codes are assigned for type:

\ /

A - ASCII | | N - Non-print

|-><-| T - Telnet format effectors

E - EBCDIC| | C - Carriage Control (ASA)

/ \

I - Image

L <byte size> - Local byte Byte size

Postel & Reynolds [Page 28]

RFC 959 October 1985

File Transfer Protocol

The default representation type is ASCII Non-print. If the

Format parameter is changed, and later just the first

argument is changed, Format then returns to the Non-print

default.

FILE STRUCTURE (STRU)

The argument is a single Telnet character code specifying

file structure described in the Section on Data

Representation and Storage.

The following codes are assigned for structure:

F - File (no record structure)

R - Record structure

P - Page structure

The default structure is File.

TRANSFER MODE (MODE)

The argument is a single Telnet character code specifying

the data transfer modes described in the Section on

Transmission Modes.

The following codes are assigned for transfer modes:

S - Stream

B - Block

C - Compressed

The default transfer mode is Stream.

4.1.3. FTP SERVICE COMMANDS

The FTP service commands define the file transfer or the file

system function requested by the user. The argument of an FTP

service command will normally be a pathname. The syntax of

pathnames must conform to server site conventions (with

standard defaults applicable), and the language conventions of

the control connection. The suggested default handling is to

use the last specified device, directory or file name, or the

standard default defined for local users. The commands may be

in any order except that a "rename from" command must be

followed by a "rename to" command and the restart command must

be followed by the interrupted service command (e.g., STOR or

RETR). The data, when transferred in response to FTP service

Postel & Reynolds [Page 29]

RFC 959 October 1985

File Transfer Protocol

commands, shall always be sent over the data connection, except

for certain informative replies. The following commands

specify FTP service requests:

RETRIEVE (RETR)

This command causes the server-DTP to transfer a copy of the

file, specified in the pathname, to the server- or user-DTP

at the other end of the data connection. The status and

contents of the file at the server site shall be unaffected.

STORE (STOR)

This command causes the server-DTP to accept the data

transferred via the data connection and to store the data as

a file at the server site. If the file specified in the

pathname exists at the server site, then its contents shall

be replaced by the data being transferred. A new file is

created at the server site if the file specified in the

pathname does not already exist.

STORE UNIQUE (STOU)

This command behaves like STOR except that the resultant

file is to be created in the current directory under a name

unique to that directory. The 250 Transfer Started response

must include the name generated.

APPEND (with create) (APPE)

This command causes the server-DTP to accept the data

transferred via the data connection and to store the data in

a file at the server site. If the file specified in the

pathname exists at the server site, then the data shall be

appended to that file; otherwise the file specified in the

pathname shall be created at the server site.

ALLOCATE (ALLO)

This command may be required by some servers to reserve

sufficient storage to accommodate the new file to be

transferred. The argument shall be a decimal integer

representing the number of bytes (using the logical byte

size) of storage to be reserved for the file. For files

sent with record or page structure a maximum record or page

size (in logical bytes) might also be necessary; this is

indicated by a decimal integer in a second argument field of

Postel & Reynolds [Page 30]

RFC 959 October 1985

File Transfer Protocol

the command. This second argument is optional, but when

present should be separated from the first by the three

Telnet characters <SP> R <SP>. This command shall be

followed by a STORe or APPEnd command. The ALLO command

should be treated as a NOOP (no operation) by those servers

which do not require that the maximum size of the file be

declared beforehand, and those servers interested in only

the maximum record or page size should accept a dummy value

in the first argument and ignore it.

RESTART (REST)

The argument field represents the server marker at which

file transfer is to be restarted. This command does not

cause file transfer but skips over the file to the specified

data checkpoint. This command shall be immediately followed

by the appropriate FTP service command which shall cause

file transfer to resume.

RENAME FROM (RNFR)

This command specifies the old pathname of the file which is

to be renamed. This command must be immediately followed by

a "rename to" command specifying the new file pathname.

RENAME TO (RNTO)

This command specifies the new pathname of the file

specified in the immediately preceding "rename from"

command. Together the two commands cause a file to be

renamed.

ABORT (ABOR)

This command tells the server to abort the previous FTP

service command and any associated transfer of data. The

abort command may require "special action", as discussed in

the Section on FTP Commands, to force recognition by the

server. No action is to be taken if the previous command

has been completed (including data transfer). The control

connection is not to be closed by the server, but the data

connection must be closed.

There are two cases for the server upon receipt of this

command: (1) the FTP service command was already completed,

or (2) the FTP service command is still in progress.

Postel & Reynolds [Page 31]

RFC 959 October 1985

File Transfer Protocol

In the first case, the server closes the data connection

(if it is open) and responds with a 226 reply, indicating

that the abort command was successfully processed.

In the second case, the server aborts the FTP service in

progress and closes the data connection, returning a 426

reply to indicate that the service request terminated

abnormally. The server then sends a 226 reply,

indicating that the abort command was successfully

processed.

DELETE (DELE)

This command causes the file specified in the pathname to be

deleted at the server site. If an extra level of protection

is desired (such as the query, "Do you really wish to

delete?"), it should be provided by the user-FTP process.

REMOVE DIRECTORY (RMD)

This command causes the directory specified in the pathname

to be removed as a directory (if the pathname is absolute)

or as a subdirectory of the current working directory (if

the pathname is relative). See Appendix II.

MAKE DIRECTORY (MKD)

This command causes the directory specified in the pathname

to be created as a directory (if the pathname is absolute)

or as a subdirectory of the current working directory (if

the pathname is relative). See Appendix II.

PRINT WORKING DIRECTORY (PWD)

This command causes the name of the current working

directory to be returned in the reply. See Appendix II.

LIST (LIST)

This command causes a list to be sent from the server to the

passive DTP. If the pathname specifies a directory or other

group of files, the server should transfer a list of files

in the specified directory. If the pathname specifies a

file then the server should send current information on the

file. A null argument implies the user's current working or

default directory. The data transfer is over the data

connection in type ASCII or type EBCDIC. (The user must

Postel & Reynolds [Page 32]

RFC 959 October 1985

File Transfer Protocol

ensure that the TYPE is appropriately ASCII or EBCDIC).

Since the information on a file may vary widely from system

to system, this information may be hard to use automatically

in a program, but may be quite useful to a human user.

NAME LIST (NLST)

This command causes a directory listing to be sent from

server to user site. The pathname should specify a

directory or other system-specific file group descriptor; a

null argument implies the current directory. The server

will return a stream of names of files and no other

information. The data will be transferred in ASCII or

EBCDIC type over the data connection as valid pathname

strings separated by <CRLF> or <NL>. (Again the user must

ensure that the TYPE is correct.) This command is intended

to return information that can be used by a program to

further process the files automatically. For example, in

the implementation of a "multiple get" function.

SITE PARAMETERS (SITE)

This command is used by the server to provide services

specific to his system that are essential to file transfer

but not sufficiently universal to be included as commands in

the protocol. The nature of these services and the

specification of their syntax can be stated in a reply to

the HELP SITE command.

SYSTEM (SYST)

This command is used to find out the type of operating

system at the server. The reply shall have as its first

word one of the system names listed in the current version

of the Assigned Numbers document [4].

STATUS (STAT)

This command shall cause a status response to be sent over

the control connection in the form of a reply. The command

may be sent during a file transfer (along with the Telnet IP

and Synch signals--see the Section on FTP Commands) in which

case the server will respond with the status of the

operation in progress, or it may be sent between file

transfers. In the latter case, the command may have an

argument field. If the argument is a pathname, the command

is analogous to the "list" command except that data shall be

Postel & Reynolds [Page 33]

RFC 959 October 1985

File Transfer Protocol

transferred over the control connection. If a partial

pathname is given, the server may respond with a list of

file names or attributes associated with that specification.

If no argument is given, the server should return general

status information about the server FTP process. This

should include current values of all transfer parameters and

the status of connections.

HELP (HELP)

This command shall cause the server to send helpful

information regarding its implementation status over the

control connection to the user. The command may take an

argument (e.g., any command name) and return more specific

information as a response. The reply is type 211 or 214.

It is suggested that HELP be allowed before entering a USER

command. The server may use this reply to specify

site-dependent parameters, e.g., in response to HELP SITE.

NOOP (NOOP)

This command does not affect any parameters or previously

entered commands. It specifies no action other than that the

server send an OK reply.

The File Transfer Protocol follows the specifications of the Telnet

protocol for all communications over the control connection. Since

the language used for Telnet communication may be a negotiated

option, all references in the next two sections will be to the

"Telnet language" and the corresponding "Telnet end-of-line code".

Currently, one may take these to mean NVT-ASCII and <CRLF>. No other

specifications of the Telnet protocol will be cited.

FTP commands are "Telnet strings" terminated by the "Telnet end of

line code". The command codes themselves are alphabetic characters

terminated by the character <SP> (Space) if parameters follow and

Telnet-EOL otherwise. The command codes and the semantics of

commands are described in this section; the detailed syntax of

commands is specified in the Section on Commands, the reply sequences

are discussed in the Section on Sequencing of Commands and Replies,

and scenarios illustrating the use of commands are provided in the

Section on Typical FTP Scenarios.

FTP commands may be partitioned as those specifying access-control

identifiers, data transfer parameters, or FTP service requests.

Certain commands (such as ABOR, STAT, QUIT) may be sent over the

control connection while a data transfer is in progress. Some

Postel & Reynolds [Page 34]

RFC 959 October 1985

File Transfer Protocol

servers may not be able to monitor the control and data connections

simultaneously, in which case some special action will be necessary

to get the server's attention. The following ordered format is

tentatively recommended:

1. User system inserts the Telnet "Interrupt Process" (IP) signal

in the Telnet stream.

2. User system sends the Telnet "Synch" signal.

3. User system inserts the command (e.g., ABOR) in the Telnet

stream.

4. Server PI, after receiving "IP", scans the Telnet stream for

EXACTLY ONE FTP command.

(For other servers this may not be necessary but the actions listed

above should have no unusual effect.)

4.2. FTP REPLIES

Replies to File Transfer Protocol commands are devised to ensure

the synchronization of requests and actions in the process of file

transfer, and to guarantee that the user process always knows the

state of the Server. Every command must generate at least one

reply, although there may be more than one; in the latter case,

the multiple replies must be easily distinguished. In addition,

some commands occur in sequential groups, such as USER, PASS and

ACCT, or RNFR and RNTO. The replies show the existence of an

intermediate state if all preceding commands have been successful.

A failure at any point in the sequence necessitates the repetition

of the entire sequence from the beginning.

The details of the command-reply sequence are made explicit in

a set of state diagrams below.

An FTP reply consists of a three digit number (transmitted as

three alphanumeric characters) followed by some text. The number

is intended for use by automata to determine what state to enter

next; the text is intended for the human user. It is intended

that the three digits contain enough encoded information that the

user-process (the User-PI) will not need to examine the text and

may either discard it or pass it on to the user, as appropriate.

In particular, the text may be server-dependent, so there are

likely to be varying texts for each reply code.

A reply is defined to contain the 3-digit code, followed by Space

Postel & Reynolds [Page 35]

RFC 959 October 1985

File Transfer Protocol

<SP>, followed by one line of text (where some maximum line length

has been specified), and terminated by the Telnet end-of-line

code. There will be cases however, where the text is longer than

a single line. In these cases the complete text must be bracketed

so the User-process knows when it may stop reading the reply (i.e.

stop processing input on the control connection) and go do other

things. This requires a special format on the first line to

indicate that more than one line is coming, and another on the

last line to designate it as the last. At least one of these must

contain the appropriate reply code to indicate the state of the

transaction. To satisfy all factions, it was decided that both

the first and last line codes should be the same.

Thus the format for multi-line replies is that the first line

will begin with the exact required reply code, followed

immediately by a Hyphen, "-" (also known as Minus), followed by

text. The last line will begin with the same code, followed

immediately by Space <SP>, optionally some text, and the Telnet

end-of-line code.

For example:

123-First line

Second line

234 A line beginning with numbers

123 The last line

The user-process then simply needs to search for the second

occurrence of the same reply code, followed by <SP> (Space), at

the beginning of a line, and ignore all intermediary lines. If

an intermediary line begins with a 3-digit number, the Server

must pad the front to avoid confusion.

This scheme allows standard system routines to be used for

reply information (such as for the STAT reply), with

"artificial" first and last lines tacked on. In rare cases

where these routines are able to generate three digits and a

Space at the beginning of any line, the beginning of each

text line should be offset by some neutral text, like Space.

This scheme assumes that multi-line replies may not be nested.

The three digits of the reply each have a special significance.

This is intended to allow a range of very simple to very

sophisticated responses by the user-process. The first digit

denotes whether the response is good, bad or incomplete.

(Referring to the state diagram), an unsophisticated user-process

will be able to determine its next action (proceed as planned,

Postel & Reynolds [Page 36]

RFC 959 October 1985

File Transfer Protocol

redo, retrench, etc.) by simply examining this first digit. A

user-process that wants to know approximately what kind of error

occurred (e.g. file system error, command syntax error) may

examine the second digit, reserving the third digit for the finest

gradation of information (e.g., RNTO command without a preceding

RNFR).

There are five values for the first digit of the reply code:

1yz Positive Preliminary reply

The requested action is being initiated; expect another

reply before proceeding with a new command. (The

user-process sending another command before the

completion reply would be in violation of protocol; but

server-FTP processes should queue any commands that

arrive while a preceding command is in progress.) This

type of reply can be used to indicate that the command

was accepted and the user-process may now pay attention

to the data connections, for implementations where

simultaneous monitoring is difficult. The server-FTP

process may send at most, one 1yz reply per command.

2yz Positive Completion reply

The requested action has been successfully completed. A

new request may be initiated.

3yz Positive Intermediate reply

The command has been accepted, but the requested action

is being held in abeyance, pending receipt of further

information. The user should send another command

specifying this information. This reply is used in

command sequence groups.

4yz Transient Negative Completion reply

The command was not accepted and the requested action did

not take place, but the error condition is temporary and

the action may be requested again. The user should

return to the beginning of the command sequence, if any.

It is difficult to assign a meaning to "transient",

particularly when two distinct sites (Server- and

User-processes) have to agree on the interpretation.

Each reply in the 4yz category might have a slightly

different time value, but the intent is that the

Postel & Reynolds [Page 37]

RFC 959 October 1985

File Transfer Protocol

user-process is encouraged to try again. A rule of thumb

in determining if a reply fits into the 4yz or the 5yz

(Permanent Negative) category is that replies are 4yz if

the commands can be repeated without any change in

command form or in properties of the User or Server

(e.g., the command is spelled the same with the same

arguments used; the user does not change his file access

or user name; the server does not put up a new

implementation.)

5yz Permanent Negative Completion reply

The command was not accepted and the requested action did

not take place. The User-process is discouraged from

repeating the exact request (in the same sequence). Even

some "permanent" error conditions can be corrected, so

the human user may want to direct his User-process to

reinitiate the command sequence by direct action at some

point in the future (e.g., after the spelling has been

changed, or the user has altered his directory status.)

The following function groupings are encoded in the second

digit:

x0z Syntax - These replies refer to syntax errors,

syntactically correct commands that don't fit any

functional category, unimplemented or superfluous

commands.

x1z Information - These are replies to requests for

information, such as status or help.

x2z Connections - Replies referring to the control and

data connections.

x3z Authentication and accounting - Replies for the login

process and accounting procedures.

x4z Unspecified as yet.

x5z File system - These replies indicate the status of the

Server file system vis-a-vis the requested transfer or

other file system action.

The third digit gives a finer gradation of meaning in each of

the function categories, specified by the second digit. The

list of replies below will illustrate this. Note that the text

Postel & Reynolds [Page 38]

RFC 959 October 1985

File Transfer Protocol

associated with each reply is recommended, rather than

mandatory, and may even change according to the command with

which it is associated. The reply codes, on the other hand,

must strictly follow the specifications in the last section;

that is, Server implementations should not invent new codes for

situations that are only slightly different from the ones

described here, but rather should adapt codes already defined.

A command such as TYPE or ALLO whose successful execution

does not offer the user-process any new information will

cause a 200 reply to be returned. If the command is not

implemented by a particular Server-FTP process because it

has no relevance to that computer system, for example ALLO

at a TOPS20 site, a Positive Completion reply is still

desired so that the simple User-process knows it can proceed

with its course of action. A 202 reply is used in this case

with, for example, the reply text: "No storage allocation

necessary." If, on the other hand, the command requests a

non-site-specific action and is unimplemented, the response

is 502. A refinement of that is the 504 reply for a command

that is implemented, but that requests an unimplemented

parameter.

4.2.1 Reply Codes by Function Groups

200 Command okay.

500 Syntax error, command unrecognized.

This may include errors such as command line too long.

501 Syntax error in parameters or arguments.

202 Command not implemented, superfluous at this site.

502 Command not implemented.

503 Bad sequence of commands.

504 Command not implemented for that parameter.

Postel & Reynolds [Page 39]

RFC 959 October 1985

File Transfer Protocol

110 Restart marker reply.

In this case, the text is exact and not left to the

particular implementation; it must read:

MARK yyyy = mmmm

Where yyyy is User-process data stream marker, and mmmm

server's equivalent marker (note the spaces between markers

and "=").

211 System status, or system help reply.

212 Directory status.

213 File status.

214 Help message.

On how to use the server or the meaning of a particular

non-standard command. This reply is useful only to the

human user.

215 NAME system type.

Where NAME is an official system name from the list in the

Assigned Numbers document.

120 Service ready in nnn minutes.

220 Service ready for new user.

221 Service closing control connection.

Logged out if appropriate.

421 Service not available, closing control connection.

This may be a reply to any command if the service knows it

must shut down.

125 Data connection already open; transfer starting.

225 Data connection open; no transfer in progress.

425 Can't open data connection.

226 Closing data connection.

Requested file action successful (for example, file

transfer or file abort).

426 Connection closed; transfer aborted.

227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).

230 User logged in, proceed.

530 Not logged in.

331 User name okay, need password.

332 Need account for login.

532 Need account for storing files.

Postel & Reynolds [Page 40]

RFC 959 October 1985

File Transfer Protocol

150 File status okay; about to open data connection.

250 Requested file action okay, completed.

257 "PATHNAME" created.

350 Requested file action pending further information.

450 Requested file action not taken.

File unavailable (e.g., file busy).

550 Requested action not taken.

File unavailable (e.g., file not found, no access).

451 Requested action aborted. Local error in processing.

551 Requested action aborted. Page type unknown.

452 Requested action not taken.

Insufficient storage space in system.

552 Requested file action aborted.

Exceeded storage allocation (for current directory or

dataset).

553 Requested action not taken.

File name not allowed.

4.2.2 Numeric Order List of Reply Codes

110 Restart marker reply.

In this case, the text is exact and not left to the

particular implementation; it must read:

MARK yyyy = mmmm

Where yyyy is User-process data stream marker, and mmmm

server's equivalent marker (note the spaces between markers

and "=").

120 Service ready in nnn minutes.

125 Data connection already open; transfer starting.

150 File status okay; about to open data connection.

Postel & Reynolds [Page 41]

RFC 959 October 1985

File Transfer Protocol

200 Command okay.

202 Command not implemented, superfluous at this site.

211 System status, or system help reply.

212 Directory status.

213 File status.

214 Help message.

On how to use the server or the meaning of a particular

non-standard command. This reply is useful only to the

human user.

215 NAME system type.

Where NAME is an official system name from the list in the

Assigned Numbers document.

220 Service ready for new user.

221 Service closing control connection.

Logged out if appropriate.

225 Data connection open; no transfer in progress.

226 Closing data connection.

Requested file action successful (for example, file

transfer or file abort).

227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).

230 User logged in, proceed.

250 Requested file action okay, completed.

257 "PATHNAME" created.

331 User name okay, need password.

332 Need account for login.

350 Requested file action pending further information.

421 Service not available, closing control connection.

This may be a reply to any command if the service knows it

must shut down.

425 Can't open data connection.

426 Connection closed; transfer aborted.

450 Requested file action not taken.

File unavailable (e.g., file busy).

451 Requested action aborted: local error in processing.

452 Requested action not taken.

Insufficient storage space in system.

Postel & Reynolds [Page 42]

RFC 959 October 1985

File Transfer Protocol

500 Syntax error, command unrecognized.

This may include errors such as command line too long.

501 Syntax error in parameters or arguments.

502 Command not implemented.

503 Bad sequence of commands.

504 Command not implemented for that parameter.

530 Not logged in.

532 Need account for storing files.

550 Requested action not taken.

File unavailable (e.g., file not found, no access).

551 Requested action aborted: page type unknown.

552 Requested file action aborted.

Exceeded storage allocation (for current directory or

dataset).

553 Requested action not taken.

File name not allowed.

5. DECLARATIVE SPECIFICATIONS

5.1. MINIMUM IMPLEMENTATION

In order to make FTP workable without needless error messages, the

following minimum implementation is required for all servers:

TYPE - ASCII Non-print

MODE - Stream

STRUCTURE - File, Record

COMMANDS - USER, QUIT, PORT,

TYPE, MODE, STRU,

for the default values

RETR, STOR,

NOOP.

The default values for transfer parameters are:

TYPE - ASCII Non-print

MODE - Stream

STRU - File

All hosts must accept the above as the standard defaults.

Postel & Reynolds [Page 43]

RFC 959 October 1985

File Transfer Protocol

5.2. CONNECTIONS

The server protocol interpreter shall "listen" on Port L. The

user or user protocol interpreter shall initiate the full-duplex

control connection. Server- and user- processes should follow the

conventions of the Telnet protocol as specified in the

ARPA-Internet Protocol Handbook [1]. Servers are under no

obligation to provide for editing of command lines and may require

that it be done in the user host. The control connection shall be

closed by the server at the user's request after all transfers and

replies are completed.

The user-DTP must "listen" on the specified data port; this may be

the default user port (U) or a port specified in the PORT command.

The server shall initiate the data connection from his own default

data port (L-1) using the specified user data port. The direction

of the transfer and the port used will be determined by the FTP

service command.

Note that all FTP implementation must support data transfer using

the default port, and that only the USER-PI may initiate the use

of non-default ports.

When data is to be transferred between two servers, A and B (refer

to Figure 2), the user-PI, C, sets up control connections with

both server-PI's. One of the servers, say A, is then sent a PASV

command telling him to "listen" on his data port rather than

initiate a connection when he receives a transfer service command.

When the user-PI receives an acknowledgment to the PASV command,

which includes the identity of the host and port being listened

on, the user-PI then sends A's port, a, to B in a PORT command; a

reply is returned. The user-PI may then send the corresponding

service commands to A and B. Server B initiates the connection

and the transfer proceeds. The command-reply sequence is listed

below where the messages are vertically synchronous but

horizontally asynchronous:

Postel & Reynolds [Page 44]

RFC 959 October 1985

File Transfer Protocol

User-PI - Server A User-PI - Server B

------------------ ------------------

C->A : Connect C->B : Connect

C->A : PASV

A->C : 227 Entering Passive Mode. A1,A2,A3,A4,a1,a2

C->B : PORT A1,A2,A3,A4,a1,a2

B->C : 200 Okay

C->A : STOR C->B : RETR

B->A : Connect to HOST-A, PORT-a

Figure 3

The data connection shall be closed by the server under the

conditions described in the Section on Establishing Data

Connections. If the data connection is to be closed following a

data transfer where closing the connection is not required to

indicate the end-of-file, the server must do so immediately.

Waiting until after a new transfer command is not permitted

because the user-process will have already tested the data

connection to see if it needs to do a "listen"; (remember that the

user must "listen" on a closed data port BEFORE sending the

transfer request). To prevent a race condition here, the server

sends a reply (226) after closing the data connection (or if the

connection is left open, a "file transfer completed" reply (250)

and the user-PI should wait for one of these replies before

issuing a new transfer command).

Any time either the user or server see that the connection is

being closed by the other side, it should promptly read any

remaining data queued on the connection and issue the close on its

own side.

5.3. COMMANDS

The commands are Telnet character strings transmitted over the

control connections as described in the Section on FTP Commands.

The command functions and semantics are described in the Section

on Access Control Commands, Transfer Parameter Commands, FTP

Service Commands, and Miscellaneous Commands. The command syntax

is specified here.

The commands begin with a command code followed by an argument

field. The command codes are four or fewer alphabetic characters.

Upper and lower case alphabetic characters are to be treated

identically. Thus, any of the following may represent the

retrieve command:

Postel & Reynolds [Page 45]

RFC 959 October 1985

File Transfer Protocol

RETR Retr retr ReTr rETr

This also applies to any symbols representing parameter values,

such as A or a for ASCII TYPE. The command codes and the argument

fields are separated by one or more spaces.

The argument field consists of a variable length character string

ending with the character sequence <CRLF> (Carriage Return, Line

Feed) for NVT-ASCII representation; for other negotiated languages

a different end of line character might be used. It should be

noted that the server is to take no action until the end of line

code is received.

The syntax is specified below in NVT-ASCII. All characters in the

argument field are ASCII characters including any ASCII

represented decimal integers. Square brackets denote an optional

argument field. If the option is not taken, the appropriate

default is implied.

Postel & Reynolds [Page 46]

RFC 959 October 1985

File Transfer Protocol

5.3.1. FTP COMMANDS

The following are the FTP commands:

USER <SP> <username> <CRLF>

PASS <SP> <password> <CRLF>

ACCT <SP> <account-information> <CRLF>

CWD <SP> <pathname> <CRLF>

CDUP <CRLF>

SMNT <SP> <pathname> <CRLF>

QUIT <CRLF>

REIN <CRLF>

PORT <SP> <host-port> <CRLF>

PASV <CRLF>

TYPE <SP> <type-code> <CRLF>

STRU <SP> <structure-code> <CRLF>

MODE <SP> <mode-code> <CRLF>

RETR <SP> <pathname> <CRLF>

STOR <SP> <pathname> <CRLF>

STOU <CRLF>

APPE <SP> <pathname> <CRLF>

ALLO <SP> <decimal-integer>

[<SP> R <SP> <decimal-integer>] <CRLF>

REST <SP> <marker> <CRLF>

RNFR <SP> <pathname> <CRLF>

RNTO <SP> <pathname> <CRLF>

ABOR <CRLF>

DELE <SP> <pathname> <CRLF>

RMD <SP> <pathname> <CRLF>

MKD <SP> <pathname> <CRLF>

PWD <CRLF>

LIST [<SP> <pathname>] <CRLF>

NLST [<SP> <pathname>] <CRLF>

SITE <SP> <string> <CRLF>

SYST <CRLF>

STAT [<SP> <pathname>] <CRLF>

HELP [<SP> <string>] <CRLF>

NOOP <CRLF>

Postel & Reynolds [Page 47]

RFC 959 October 1985

File Transfer Protocol

5.3.2. FTP COMMAND ARGUMENTS

The syntax of the above argument fields (using BNF notation

where applicable) is:

<username> ::= <string>

<password> ::= <string>

<account-information> ::= <string>

<string> ::= <char> | <char><string>

<char> ::= any of the 128 ASCII characters except <CR> and

<LF>

<marker> ::= <pr-string>

<pr-string> ::= <pr-char> | <pr-char><pr-string>

<pr-char> ::= printable characters, any

ASCII code 33 through 126

<byte-size> ::= <number>

<host-port> ::= <host-number>,<port-number>

<host-number> ::= <number>,<number>,<number>,<number>

<port-number> ::= <number>,<number>

<number> ::= any decimal integer 1 through 255

<form-code> ::= N | T | C

<type-code> ::= A [<sp> <form-code>]

| E [<sp> <form-code>]

| I

| L <sp> <byte-size>

<structure-code> ::= F | R | P

<mode-code> ::= S | B | C

<pathname> ::= <string>

<decimal-integer> ::= any decimal integer

Postel & Reynolds [Page 48]

RFC 959 October 1985

File Transfer Protocol

5.4. SEQUENCING OF COMMANDS AND REPLIES

The communication between the user and server is intended to be an

alternating dialogue. As such, the user issues an FTP command and

the server responds with a prompt primary reply. The user should

wait for this initial primary success or failure response before

sending further commands.

Certain commands require a second reply for which the user should

also wait. These replies may, for example, report on the progress

or completion of file transfer or the closing of the data

connection. They are secondary replies to file transfer commands.

One important group of informational replies is the connection

greetings. Under normal circumstances, a server will send a 220

reply, "awaiting input", when the connection is completed. The

user should wait for this greeting message before sending any

commands. If the server is unable to accept input right away, a

120 "expected delay" reply should be sent immediately and a 220

reply when ready. The user will then know not to hang up if there

is a delay.

Spontaneous Replies

Sometimes "the system" spontaneously has a message to be sent

to a user (usually all users). For example, "System going down

in 15 minutes". There is no provision in FTP for such

spontaneous information to be sent from the server to the user.

It is recommended that such information be queued in the

server-PI and delivered to the user-PI in the next reply

(possibly making it a multi-line reply).

The table below lists alternative success and failure replies for

each command. These must be strictly adhered to; a server may

substitute text in the replies, but the meaning and action implied

by the code numbers and by the specific command reply sequence

cannot be altered.

Command-Reply Sequences

In this section, the command-reply sequence is presented. Each

command is listed with its possible replies; command groups are

listed together. Preliminary replies are listed first (with

their succeeding replies indented and under them), then

positive and negative completion, and finally intermediary

Postel & Reynolds [Page 49]

RFC 959 October 1985

File Transfer Protocol

replies with the remaining commands from the sequence

following. This listing forms the basis for the state

diagrams, which will be presented separately.

Connection Establishment

120

220

220

421

Login

USER

230

530

500, 501, 421

331, 332

PASS

230

202

530

500, 501, 503, 421

332

ACCT

230

202

530

500, 501, 503, 421

CWD

250

500, 501, 502, 421, 530, 550

CDUP

200

500, 501, 502, 421, 530, 550

SMNT

202, 250

500, 501, 502, 421, 530, 550

Logout

REIN

120

220

220

421

500, 502

QUIT

221

500

Postel & Reynolds [Page 50]

RFC 959 October 1985

File Transfer Protocol

Transfer parameters

PORT

200

500, 501, 421, 530

PASV

227

500, 501, 502, 421, 530

MODE

200

500, 501, 504, 421, 530

TYPE

200

500, 501, 504, 421, 530

STRU

200

500, 501, 504, 421, 530

File action commands

ALLO

200

202

500, 501, 504, 421, 530

REST

500, 501, 502, 421, 530

350

STOR

125, 150

(110)

226, 250

425, 426, 451, 551, 552

532, 450, 452, 553

500, 501, 421, 530

STOU

125, 150

(110)

226, 250

425, 426, 451, 551, 552

532, 450, 452, 553

500, 501, 421, 530

RETR

125, 150

(110)

226, 250

425, 426, 451

450, 550

500, 501, 421, 530

Postel & Reynolds [Page 51]

RFC 959 October 1985

File Transfer Protocol

LIST

125, 150

226, 250

425, 426, 451

450

500, 501, 502, 421, 530

NLST

125, 150

226, 250

425, 426, 451

450

500, 501, 502, 421, 530

APPE

125, 150

(110)

226, 250

425, 426, 451, 551, 552

532, 450, 550, 452, 553

500, 501, 502, 421, 530

RNFR

450, 550

500, 501, 502, 421, 530

350

RNTO

250

532, 553

500, 501, 502, 503, 421, 530

DELE

250

450, 550

500, 501, 502, 421, 530

RMD

250

500, 501, 502, 421, 530, 550

MKD

257

500, 501, 502, 421, 530, 550

PWD

257

500, 501, 502, 421, 550

ABOR

225, 226

500, 501, 502, 421

Postel & Reynolds [Page 52]

RFC 959 October 1985

File Transfer Protocol

Informational commands

SYST

215

500, 501, 502, 421

STAT

211, 212, 213

450

500, 501, 502, 421, 530

HELP

211, 214

500, 501, 502, 421

Miscellaneous commands

SITE

200

202

500, 501, 530

NOOP

200

500 421

Postel & Reynolds [Page 53]

RFC 959 October 1985

File Transfer Protocol

6. STATE DIAGRAMS

Here we present state diagrams for a very simple minded FTP

implementation. Only the first digit of the reply codes is used.

There is one state diagram for each group of FTP commands or command

sequences.

The command groupings were determined by constructing a model for

each command then collecting together the commands with structurally

identical models.

For each command or command sequence there are three possible

outcomes: success (S), failure (F), and error (E). In the state

diagrams below we use the symbol B for "begin", and the symbol W for

"wait for reply".

We first present the diagram that represents the largest group of FTP

commands:

1,3 +---+

----------->| E |

| +---+

|

+---+ cmd +---+ 2 +---+

| B |---------->| W |---------->| S |

+---+ +---+ +---+

|

| 4,5 +---+

----------->| F |

+---+

This diagram models the commands:

ABOR, ALLO, DELE, CWD, CDUP, SMNT, HELP, MODE, NOOP, PASV,

QUIT, SITE, PORT, SYST, STAT, RMD, MKD, PWD, STRU, and TYPE.

Postel & Reynolds [Page 54]

RFC 959 October 1985

File Transfer Protocol

The other large group of commands is represented by a very similar

diagram:

3 +---+

----------->| E |

| +---+

|

+---+ cmd +---+ 2 +---+

| B |---------->| W |---------->| S |

+---+ --->+---+ +---+

| | |

| | | 4,5 +---+

| 1 | ----------->| F |

----- +---+

This diagram models the commands:

APPE, LIST, NLST, REIN, RETR, STOR, and STOU.

Note that this second model could also be used to represent the first

group of commands, the only difference being that in the first group

the 100 series replies are unexpected and therefore treated as error,

while the second group expects (some may require) 100 series replies.

Remember that at most, one 100 series reply is allowed per command.

The remaining diagrams model command sequences, perhaps the simplest

of these is the rename sequence:

+---+ RNFR +---+ 1,2 +---+

| B |---------->| W |---------->| E |

+---+ +---+ -->+---+

| | |

3 | | 4,5 |

-------------- ------ |

| | | +---+

| ------------->| S |

| | 1,3 | | +---+

| 2| --------

| | | |

V | | |

+---+ RNTO +---+ 4,5 ----->+---+

| |---------->| W |---------->| F |

+---+ +---+ +---+

Postel & Reynolds [Page 55]

RFC 959 October 1985

File Transfer Protocol

The next diagram is a simple model of the Restart command:

+---+ REST +---+ 1,2 +---+

| B |---------->| W |---------->| E |

+---+ +---+ -->+---+

| | |

3 | | 4,5 |

-------------- ------ |

| | | +---+

| ------------->| S |

| | 3 | | +---+

| 2| --------

| | | |

V | | |

+---+ cmd +---+ 4,5 ----->+---+

| |---------->| W |---------->| F |

+---+ -->+---+ +---+

| |

| 1 |

------

Where "cmd" is APPE, STOR, or RETR.

We note that the above three models are similar. The Restart differs

from the Rename two only in the treatment of 100 series replies at

the second stage, while the second group expects (some may require)

100 series replies. Remember that at most, one 100 series reply is

allowed per command.

Postel & Reynolds [Page 56]

RFC 959 October 1985

File Transfer Protocol

The most complicated diagram is for the Login sequence:

1

+---+ USER +---+------------->+---+

| B |---------->| W | 2 ---->| E |

+---+ +---+------ | -->+---+

| | | | |

3 | | 4,5 | | |

-------------- ----- | | |

| | | | |

| | | | |

| --------- |

| 1| | | |

V | | | |

+---+ PASS +---+ 2 | ------>+---+

| |---------->| W |------------->| S |

+---+ +---+ ---------->+---+

| | | | |

3 | |4,5| | |

-------------- -------- |

| | | | |

| | | | |

| -----------

| 1,3| | | |

V | 2| | |

+---+ ACCT +---+-- | ----->+---+

| |---------->| W | 4,5 -------->| F |

+---+ +---+------------->+---+

Postel & Reynolds [Page 57]

RFC 959 October 1985

File Transfer Protocol

Finally, we present a generalized diagram that could be used to model

the command and reply interchange:

------------------------------------

| |

Begin | |

| V |

| +---+ cmd +---+ 2 +---+ |

-->| |------->| |---------->| | |

| | | W | | S |-----|

-->| | -->| |----- | | |

| +---+ | +---+ 4,5 | +---+ |

| | | | | | |

| | | 1| |3 | +---+ |

| | | | | | | | |

| | ---- | ---->| F |-----

| | | | |

| | | +---+

-------------------

|

|

V

End

Postel & Reynolds [Page 58]

RFC 959 October 1985

File Transfer Protocol

7. TYPICAL FTP SCENARIO

User at host U wanting to transfer files to/from host S:

In general, the user will communicate to the server via a mediating

user-FTP process. The following may be a typical scenario. The

user-FTP prompts are shown in parentheses, '---->' represents

commands from host U to host S, and '<----' represents replies from

host S to host U.

LOCAL COMMANDS BY USER ACTION INVOLVED

ftp (host) multics<CR> Connect to host S, port L,

establishing control connections.

<---- 220 Service ready <CRLF>.

username Doe <CR> USER Doe<CRLF>---->

<---- 331 User name ok,

need password<CRLF>.

password mumble <CR> PASS mumble<CRLF>---->

<---- 230 User logged in<CRLF>.

retrieve (local type) ASCII<CR>

(local pathname) test 1 <CR> User-FTP opens local file in ASCII.

(for. pathname) test.pl1<CR> RETR test.pl1<CRLF> ---->

<---- 150 File status okay;

about to open data

connection<CRLF>.

Server makes data connection

to port U.

<---- 226 Closing data connection,

file transfer successful<CRLF>.

type Image<CR> TYPE I<CRLF> ---->

<---- 200 Command OK<CRLF>

store (local type) image<CR>

(local pathname) file dump<CR> User-FTP opens local file in Image.

(for.pathname) >udd>cn>fd<CR> STOR >udd>cn>fd<CRLF> ---->

<---- 550 Access denied<CRLF>

terminate QUIT <CRLF> ---->

Server closes all

connections.

8. CONNECTION ESTABLISHMENT

The FTP control connection is established via TCP between the user

process port U and the server process port L. This protocol is

assigned the service port 21 (25 octal), that is L=21.

Postel & Reynolds [Page 59]

RFC 959 October 1985

File Transfer Protocol

APPENDIX I - PAGE STRUCTURE

The need for FTP to support page structure derives principally from

the need to support efficient transmission of files between TOPS-20

systems, particularly the files used by NLS.

The file system of TOPS-20 is based on the concept of pages. The

operating system is most efficient at manipulating files as pages.

The operating system provides an interface to the file system so that

many applications view files as sequential streams of characters.

However, a few applications use the underlying page structures

directly, and some of these create holey files.

A TOPS-20 disk file consists of four things: a pathname, a page

table, a (possibly empty) set of pages, and a set of attributes.

The pathname is specified in the RETR or STOR command. It includes

the directory name, file name, file name extension, and generation

number.

The page table contains up to 2\*\*18 entries. Each entry may be

EMPTY, or may point to a page. If it is not empty, there are also

some page-specific access bits; not all pages of a file need have the

same access protection.

A page is a contiguous set of 512 words of 36 bits each.

The attributes of the file, in the File Descriptor Block (FDB),

contain such things as creation time, write time, read time, writer's

byte-size, end-of-file pointer, count of reads and writes, backup

system tape numbers, etc.

Note that there is NO requirement that entries in the page table be

contiguous. There may be empty page table slots between occupied

ones. Also, the end of file pointer is simply a number. There is no

requirement that it in fact point at the "last" datum in the file.

Ordinary sequential I/O calls in TOPS-20 will cause the end of file

pointer to be left after the last datum written, but other operations

may cause it not to be so, if a particular programming system so

requires.

In fact, in both of these special cases, "holey" files and

end-of-file pointers NOT at the end of the file, occur with NLS data

files.

Postel & Reynolds [Page 60]

RFC 959 October 1985

File Transfer Protocol

The TOPS-20 paged files can be sent with the FTP transfer parameters:

TYPE L 36, STRU P, and MODE S (in fact, any mode could be used).

Each page of information has a header. Each header field, which is a

logical byte, is a TOPS-20 word, since the TYPE is L 36.

The header fields are:

Word 0: Header Length.

The header length is 5.

Word 1: Page Index.

If the data is a disk file page, this is the number of that

page in the file's page map. Empty pages (holes) in the file

are simply not sent. Note that a hole is NOT the same as a

page of zeros.

Word 2: Data Length.

The number of data words in this page, following the header.

Thus, the total length of the transmission unit is the Header

Length plus the Data Length.

Word 3: Page Type.

A code for what type of chunk this is. A data page is type 3,

the FDB page is type 2.

Word 4: Page Access Control.

The access bits associated with the page in the file's page

map. (This full word quantity is put into AC2 of an SPACS by

the program reading from net to disk.)

After the header are Data Length data words. Data Length is

currently either 512 for a data page or 31 for an FDB. Trailing

zeros in a disk file page may be discarded, making Data Length less

than 512 in that case.

Postel & Reynolds [Page 61]

RFC 959 October 1985

File Transfer Protocol

APPENDIX II - DIRECTORY COMMANDS

Since UNIX has a tree-like directory structure in which directories

are as easy to manipulate as ordinary files, it is useful to expand

the FTP servers on these machines to include commands which deal with

the creation of directories. Since there are other hosts on the

ARPA-Internet which have tree-like directories (including TOPS-20 and

Multics), these commands are as general as possible.

Four directory commands have been added to FTP:

MKD pathname

Make a directory with the name "pathname".

RMD pathname

Remove the directory with the name "pathname".

PWD

Print the current working directory name.

CDUP

Change to the parent of the current working directory.

The "pathname" argument should be created (removed) as a

subdirectory of the current working directory, unless the "pathname"

string contains sufficient information to specify otherwise to the

server, e.g., "pathname" is an absolute pathname (in UNIX and

Multics), or pathname is something like "<abso.lute.path>" to

TOPS-20.

REPLY CODES

The CDUP command is a special case of CWD, and is included to

simplify the implementation of programs for transferring directory

trees between operating systems having different syntaxes for

naming the parent directory. The reply codes for CDUP be

identical to the reply codes of CWD.

The reply codes for RMD be identical to the reply codes for its

file analogue, DELE.

The reply codes for MKD, however, are a bit more complicated. A

freshly created directory will probably be the object of a future

Postel & Reynolds [Page 62]

RFC 959 October 1985

File Transfer Protocol

CWD command. Unfortunately, the argument to MKD may not always be

a suitable argument for CWD. This is the case, for example, when

a TOPS-20 subdirectory is created by giving just the subdirectory

name. That is, with a TOPS-20 server FTP, the command sequence

MKD MYDIR

CWD MYDIR

will fail. The new directory may only be referred to by its

"absolute" name; e.g., if the MKD command above were issued while

connected to the directory <DFRANKLIN>, the new subdirectory

could only be referred to by the name <DFRANKLIN.MYDIR>.

Even on UNIX and Multics, however, the argument given to MKD may

not be suitable. If it is a "relative" pathname (i.e., a pathname

which is interpreted relative to the current directory), the user

would need to be in the same current directory in order to reach

the subdirectory. Depending on the application, this may be

inconvenient. It is not very robust in any case.

To solve these problems, upon successful completion of an MKD

command, the server should return a line of the form:

257<space>"<directory-name>"<space><commentary>

That is, the server will tell the user what string to use when

referring to the created directory. The directory name can

contain any character; embedded double-quotes should be escaped by

double-quotes (the "quote-doubling" convention).

For example, a user connects to the directory /usr/dm, and creates

a subdirectory, named pathname:

CWD /usr/dm

200 directory changed to /usr/dm

MKD pathname

257 "/usr/dm/pathname" directory created

An example with an embedded double quote:

MKD foo"bar

257 "/usr/dm/foo""bar" directory created

CWD /usr/dm/foo"bar

200 directory changed to /usr/dm/foo"bar

Postel & Reynolds [Page 63]

RFC 959 October 1985

File Transfer Protocol

The prior existence of a subdirectory with the same name is an

error, and the server must return an "access denied" error reply

in that case.

CWD /usr/dm

200 directory changed to /usr/dm

MKD pathname

521-"/usr/dm/pathname" directory already exists;

521 taking no action.

The failure replies for MKD are analogous to its file creating

cousin, STOR. Also, an "access denied" return is given if a file

name with the same name as the subdirectory will conflict with the

creation of the subdirectory (this is a problem on UNIX, but

shouldn't be one on TOPS-20).

Essentially because the PWD command returns the same type of

information as the successful MKD command, the successful PWD

command uses the 257 reply code as well.

SUBTLETIES

Because these commands will be most useful in transferring

subtrees from one machine to another, carefully observe that the

argument to MKD is to be interpreted as a sub-directory of the

current working directory, unless it contains enough information

for the destination host to tell otherwise. A hypothetical

example of its use in the TOPS-20 world:

CWD <some.where>

200 Working directory changed

MKD overrainbow

257 "<some.where.overrainbow>" directory created

CWD overrainbow

431 No such directory

CWD <some.where.overrainbow>

200 Working directory changed

CWD <some.where>

200 Working directory changed to <some.where>

MKD <unambiguous>

257 "<unambiguous>" directory created

CWD <unambiguous>

Note that the first example results in a subdirectory of the

connected directory. In contrast, the argument in the second

example contains enough information for TOPS-20 to tell that the

Postel & Reynolds [Page 64]

RFC 959 October 1985

File Transfer Protocol

<unambiguous> directory is a top-level directory. Note also that

in the first example the user "violated" the protocol by

attempting to access the freshly created directory with a name

other than the one returned by TOPS-20. Problems could have

resulted in this case had there been an <overrainbow> directory;

this is an ambiguity inherent in some TOPS-20 implementations.

Similar considerations apply to the RMD command. The point is

this: except where to do so would violate a host's conventions for

denoting relative versus absolute pathnames, the host should treat

the operands of the MKD and RMD commands as subdirectories. The

257 reply to the MKD command must always contain the absolute

pathname of the created directory.

Postel & Reynolds [Page 65]

RFC 959 October 1985

File Transfer Protocol

APPENDIX III - RFCs on FTP

Bhushan, Abhay, "A File Transfer Protocol", RFC 114 (NIC 5823),

MIT-Project MAC, 16 April 1971.

Harslem, Eric, and John Heafner, "Comments on RFC 114 (A File

Transfer Protocol)", RFC 141 (NIC 6726), RAND, 29 April 1971.

Bhushan, Abhay, et al, "The File Transfer Protocol", RFC 172

(NIC 6794), MIT-Project MAC, 23 June 1971.

Braden, Bob, "Comments on DTP and FTP Proposals", RFC 238 (NIC 7663),

UCLA/CCN, 29 September 1971.

Bhushan, Abhay, et al, "The File Transfer Protocol", RFC 265

(NIC 7813), MIT-Project MAC, 17 November 1971.

McKenzie, Alex, "A Suggested Addition to File Transfer Protocol",

RFC 281 (NIC 8163), BBN, 8 December 1971.

Bhushan, Abhay, "The Use of "Set Data Type" Transaction in File

Transfer Protocol", RFC 294 (NIC 8304), MIT-Project MAC,

25 January 1972.

Bhushan, Abhay, "The File Transfer Protocol", RFC 354 (NIC 10596),

MIT-Project MAC, 8 July 1972.

Bhushan, Abhay, "Comments on the File Transfer Protocol (RFC 354)",

RFC 385 (NIC 11357), MIT-Project MAC, 18 August 1972.

Hicks, Greg, "User FTP Documentation", RFC 412 (NIC 12404), Utah,

27 November 1972.

Bhushan, Abhay, "File Transfer Protocol (FTP) Status and Further

Comments", RFC 414 (NIC 12406), MIT-Project MAC, 20 November 1972.

Braden, Bob, "Comments on File Transfer Protocol", RFC 430

(NIC 13299), UCLA/CCN, 7 February 1973.

Thomas, Bob, and Bob Clements, "FTP Server-Server Interaction",

RFC 438 (NIC 13770), BBN, 15 January 1973.

Braden, Bob, "Print Files in FTP", RFC 448 (NIC 13299), UCLA/CCN,

27 February 1973.

McKenzie, Alex, "File Transfer Protocol", RFC 454 (NIC 14333), BBN,

16 February 1973.

Postel & Reynolds [Page 66]

RFC 959 October 1985

File Transfer Protocol

Bressler, Bob, and Bob Thomas, "Mail Retrieval via FTP", RFC 458

(NIC 14378), BBN-NET and BBN-TENEX, 20 February 1973.

Neigus, Nancy, "File Transfer Protocol", RFC 542 (NIC 17759), BBN,

12 July 1973.

Krilanovich, Mark, and George Gregg, "Comments on the File Transfer

Protocol", RFC 607 (NIC 21255), UCSB, 7 January 1974.

Pogran, Ken, and Nancy Neigus, "Response to RFC 607 - Comments on the

File Transfer Protocol", RFC 614 (NIC 21530), BBN, 28 January 1974.

Krilanovich, Mark, George Gregg, Wayne Hathaway, and Jim White,

"Comments on the File Transfer Protocol", RFC 624 (NIC 22054), UCSB,

Ames Research Center, SRI-ARC, 28 February 1974.

Bhushan, Abhay, "FTP Comments and Response to RFC 430", RFC 463

(NIC 14573), MIT-DMCG, 21 February 1973.

Braden, Bob, "FTP Data Compression", RFC 468 (NIC 14742), UCLA/CCN,

8 March 1973.

Bhushan, Abhay, "FTP and Network Mail System", RFC 475 (NIC 14919),

MIT-DMCG, 6 March 1973.

Bressler, Bob, and Bob Thomas "FTP Server-Server Interaction - II",

RFC 478 (NIC 14947), BBN-NET and BBN-TENEX, 26 March 1973.

White, Jim, "Use of FTP by the NIC Journal", RFC 479 (NIC 14948),

SRI-ARC, 8 March 1973.

White, Jim, "Host-Dependent FTP Parameters", RFC 480 (NIC 14949),

SRI-ARC, 8 March 1973.

Padlipsky, Mike, "An FTP Command-Naming Problem", RFC 506

(NIC 16157), MIT-Multics, 26 June 1973.

Day, John, "Memo to FTP Group (Proposal for File Access Protocol)",

RFC 520 (NIC 16819), Illinois, 25 June 1973.

Merryman, Robert, "The UCSD-CC Server-FTP Facility", RFC 532

(NIC 17451), UCSD-CC, 22 June 1973.

Braden, Bob, "TENEX FTP Problem", RFC 571 (NIC 18974), UCLA/CCN,

15 November 1973.

Postel & Reynolds [Page 67]

RFC 959 October 1985

File Transfer Protocol

McKenzie, Alex, and Jon Postel, "Telnet and FTP Implementation -

Schedule Change", RFC 593 (NIC 20615), BBN and MITRE,

29 November 1973.

Sussman, Julie, "FTP Error Code Usage for More Reliable Mail

Service", RFC 630 (NIC 30237), BBN, 10 April 1974.

Postel, Jon, "Revised FTP Reply Codes", RFC 640 (NIC 30843),

UCLA/NMC, 5 June 1974.

Harvey, Brian, "Leaving Well Enough Alone", RFC 686 (NIC 32481),

SU-AI, 10 May 1975.

Harvey, Brian, "One More Try on the FTP", RFC 691 (NIC 32700), SU-AI,

28 May 1975.

Lieb, J., "CWD Command of FTP", RFC 697 (NIC 32963), 14 July 1975.

Harrenstien, Ken, "FTP Extension: XSEN", RFC 737 (NIC 42217), SRI-KL,

31 October 1977.

Harrenstien, Ken, "FTP Extension: XRSQ/XRCP", RFC 743 (NIC 42758),

SRI-KL, 30 December 1977.

Lebling, P. David, "Survey of FTP Mail and MLFL", RFC 751, MIT,

10 December 1978.

Postel, Jon, "File Transfer Protocol Specification", RFC 765, ISI,

June 1980.

Mankins, David, Dan Franklin, and Buzz Owen, "Directory Oriented FTP

Commands", RFC 776, BBN, December 1980.

Padlipsky, Michael, "FTP Unique-Named Store Command", RFC 949, MITRE,

July 1985.

Postel & Reynolds [Page 68]

RFC 959 October 1985

File Transfer Protocol

REFERENCES

[1] Feinler, Elizabeth, "Internet Protocol Transition Workbook",

Network Information Center, SRI International, March 1982.

[2] Postel, Jon, "Transmission Control Protocol - DARPA Internet

Program Protocol Specification", RFC 793, DARPA, September 1981.

[3] Postel, Jon, and Joyce Reynolds, "Telnet Protocol

Specification", RFC 854, ISI, May 1983.

[4] Reynolds, Joyce, and Jon Postel, "Assigned Numbers", RFC 943,

ISI, April 1985.

Postel & Reynolds [Page 69]