Network Working Group
Request for Comments: 3659
Updates: 959
Category: Standards Track

P. Hethmon
Hethmon Software

March 2007

Extensions to FTP

Status of This Memo

Copyright Notice

Abstract

This document specifies new FTP commands to obtain listings of remote directories in a defined format, and to permit restarts of interrupted data transfers in STREAM mode. It allows character sets other than US-ASCII, and also defines an optional virtual file storage structure.

Table of Contents

1. Introduction

This document updates the File Transfer Protocol (FTP) [3]. Four new commands are added: "SIZE", "MDTM", "MLST", and "MLSD". The existing command "REST" is modified. Of those, the "SIZE" and "MDTM" commands, and the modifications to "REST" have been in wide use for many years. The others are new.

These commands allow a client to restart an interrupted transfer in transfer modes not previously supported in any documented way, and to obtain a directory listing in a machine friendly, predictable, format.

An optional structure for the server's file store (NVFS) is also defined, allowing servers that support such a structure to convey that information to clients in a standard way, thus allowing clients more certainty in constructing and interpreting pathnames.

2. Document Conventions

This document makes use of the document conventions defined in BCP 14, RFC 2119 [4]. That provides the interpretation of capitalized imperative words like MUST, SHOULD, etc.

This document also uses notation defined in STD 9, RFC 959 [3]. In particular, the terms "reply", "user", "NVFS" (Network Virtual File System), "file", "pathname", "FTP commands", "DTP" (data transfer process), "user-FTP process", "user-PI" (user protocol interpreter), "user-DTP", "server-FTP process", "server-PI", "server-DTP", "mode", "type", "NVT" (Network Virtual Terminal), "control connection", "data connection", and "ASCII", are all used here as defined there.

Syntax required is defined using the Augmented BNF defined in [5]. Some general ABNF definitions that are required throughout the document will be defined later in this section. At first reading, it may be wise to simply recall that these definitions exist here, and skip to the next section.

2.1. Basic Tokens

This document imports the core ABNF definitions given in Appendix A of [5]. There definitions will be found for basic ABNF elements like ALPHA, DIGIT, SP, etc. The following terms are added for use in this document.

```
TCHAR          = VCHAR / SP / HTAB    ; visible plus white space
RCHAR          = ALPHA / DIGIT / "," / "." / ":" / "!" /
                 "@" / "#" / "$" / "%" / "^" /
                 "&" / "(" / ")" / "-" / "_" /
                 "+" / "?" / "/" / "\" / "'" /
                 DQUOTE   ; <"> -- double quote character (%x22)
SCHAR          = RCHAR / "=" ;
```

The VCHAR (from [5]), RCHAR, SCHAR, and TCHAR types give basic character types from varying sub-sets of the ASCII character set for use in various commands and responses.

```
token          = 1*RCHAR
```

A "token" is a string whose precise meaning depends upon the context in which it is used. In some cases it will be a value from a set of possible values maintained elsewhere. In others it might be a string invented by one party to an FTP conversation from whatever sources it finds relevant.

Note that in ABNF, string literals are case insensitive. That convention is preserved in this document, and implies that FTP commands added by this specification have names that can be represented in any case. That is, "MDTM" is the same as "mdtm", "Mdtm" and "MdTm" etc. However note that ALPHA, in particular, is case sensitive. That implies that a "token" is a case sensitive value. That implication is correct, except where explicitly stated to the contrary in this document, or in some other specification that defines the values this document specifies be used in a particular context.

2.2. Pathnames

Various FTP commands take pathnames as arguments, or return pathnames in responses. When the MLST command is supported, as indicated in the response to the FEAT command [6], pathnames are to be transferred in one of the following two formats.

```
pathname       = utf-8-name / raw
utf-8-name     = <a UTF-8 encoded Unicode string>
raw            = <any string that is not a valid UTF-8 encoding>
```

Which format is used is at the option of the user-PI or server-PI sending the pathname. UTF-8 encodings [2] contain enough internal structure that it is always, in practice, possible to determine whether a UTF-8 or raw encoding has been used, in those cases where it matters. While it is useful for the user-PI to be able to correctly display a pathname received from the server-PI to the user, it is far more important for the user-PI to be able to retain and retransmit the identical pathname when required. Implementations are advised against converting a UTF-8 pathname to a local charset that isn't capable of representing the full Unicode character repertoire, and then attempting to invert the charset translation later. Note that ASCII is a subset of UTF-8. See also [1].

Unless otherwise specified, the pathname is terminated by the CRLF that terminates the FTP command, or by the CRLF that ends a reply. Any trailing spaces preceding that CRLF form part of the name. Exactly one space will precede the pathname and serve as a separator from the preceding syntax element. Any additional spaces form part of the pathname. See [7] for a fuller explanation of the character encoding issues. All implementations supporting MLST MUST support [7].

Note: for pathnames transferred over a data connection, there is no way to represent a pathname containing the characters CR and LF in sequence, and distinguish that from the end of line indication. Hence, pathnames containing the CRLF pair of characters cannot be transmitted over a data connection. Data connections only contain file names transmitted from server-FTP to user-FTP as the result of one of the directory listing commands. Files with names containing the CRLF sequence must either have that sequence converted to some other form, such that the other form can be recognised and be correctly converted back to CRLF, or be omitted from the listing.

Implementations should also beware that the FTP control connection uses Telnet NVT conventions [8], and that the Telnet IAC character, if part of a pathname sent over the control connection, MUST be correctly escaped as defined by the Telnet protocol.

NVT also distinguishes between CR, LF, and the end of line CRLF, and so would permit pathnames containing the pair of characters CR and LF to be correctly transmitted. However, because such a sequence cannot be transmitted over a data connection (as part of the result of a LIST, NLST, or MLSD command), such pathnames are best avoided.

Implementors should also be aware that, although Telnet NVT conventions are used over the control connections, Telnet option negotiation MUST NOT be attempted. See section 4.1.2.12 of [9].

2.2.1. Pathname Syntax

Except where TVFS is supported (see section 6), this specification imposes no syntax upon pathnames. Nor does it restrict the character set from which pathnames are created. This does not imply that the NVFS is required to make sense of all possible pathnames. Server-PIs may restrict the syntax of valid pathnames in their NVFS in any manner appropriate to their implementation or underlying file system. Similarly, a server-PI may parse the pathname and assign meaning to the components detected.

2.2.2. Wildcarding

For the commands defined in this specification, all pathnames are to be treated literally. That is, for a pathname given as a parameter to a command, the file whose name is identical to the pathname given is implied. No characters from the pathname may be treated as special or "magic", thus no pattern matching (other than for exact equality) between the pathname given and the files present in the NVFS of the server-FTP is permitted.

Clients that desire some form of pattern matching functionality must obtain a listing of the relevant directory, or directories, and implement their own file name selection procedures.

2.3. Times

The syntax of a time value is:

```
time-val       = 14DIGIT [ "." 1*DIGIT ]
```

The leading, mandatory, fourteen digits are to be interpreted as, in order from the leftmost, four digits giving the year, with a range of 1000--9999, two digits giving the month of the year, with a range of 01--12, two digits giving the day of the month, with a range of 01--31, two digits giving the hour of the day, with a range of 00--23, two digits giving minutes past the hour, with a range of 00--59, and finally, two digits giving seconds past the minute, with a range of 00--60 (with 60 being used only at a leap second). Years in the tenth century, and earlier, cannot be expressed. This is not considered a serious defect of the protocol.

The optional digits, which are preceded by a period, give decimal fractions of a second. These may be given to whatever precision is appropriate to the circumstance, however implementations MUST NOT add precision to time-vals where that precision does not exist in the underlying value being transmitted.

Symbolically, a time-val may be viewed as

```
YYYYMMDDHHMMSS.sss
```

The "." and subsequent digits ("sss") are optional. However the "." MUST NOT appear unless at least one following digit also appears.

Time values are always represented in UTC (GMT), and in the Gregorian calendar regardless of what calendar may have been in use at the date and time indicated at the location of the server-PI.

The technical differences among GMT, TAI, UTC, UT1, UT2, etc., are not considered here. A server-FTP process should always use the same time reference, so the times it returns will be consistent. Clients are not expected to be time synchronized with the server, so the possible difference in times that might be reported by the different time standards is not considered important.

2.4. Server Replies

Section 4.2 of [3] defines the format and meaning of replies by the server-PI to FTP commands from the user-PI. Those reply conventions are used here without change.

```
error-response = error-code SP *TCHAR CRLF
error-code     = ("4" / "5") 2DIGIT
```

Implementors should note that the ABNF syntax used in this document and in other FTP related documents (but not used in [3]), sometimes shows replies using the one-line format. Unless otherwise explicitly stated, that is not intended to imply that multi-line responses are not permitted. Implementors should assume that, unless stated to the contrary, any reply to any FTP command (including QUIT) may use the multi-line format described in [3].

Throughout this document, replies will be identified by the three digit code that is their first element. Thus the term "500 reply" means a reply from the server-PI using the three digit code "500".

2.5. Interpreting Examples

In the examples of FTP dialogs presented in this document, lines that begin "C> " were sent over the control connection from the user-PI to the server-PI, lines that begin "S> " were sent over the control connection from the server-PI to the user-PI, and each sequence of lines that begin "D> " was sent from the server-PI to the user-PI over a data connection created just to send those lines and closed immediately after. No examples here show data transferred over a data connection from the client to the server. In all cases, the prefixes shown above, including the one space, have been added for the purposes of this document, and are not a part of the data exchanged between client and server.

3. File Modification Time (MDTM)

The FTP command, MODIFICATION TIME (MDTM), can be used to determine when a file in the server NVFS was last modified. This command has existed in many FTP servers for many years, as an adjunct to the REST command for STREAM mode, thus is widely available. However, where supported, the "modify" fact that can be provided in the result from the new MLST command is recommended as a superior alternative.

When attempting to restart a RETRieve, the user-FTP can use the MDTM command or the "modify" fact to check if the modification time of the source file is more recent than the modification time of the partially transferred file. If it is, then most likely the source file has changed, and it would be unsafe to restart the previously incomplete file transfer.

Because the user- and server-FTPs' clocks are not necessarily synchronised, user-FTPs intending to use this method should usually obtain the modification time of the file from the server before the initial RETRieval, and compare that with the modification time before a RESTart. If they differ, the files may have changed, and RESTart would be inadvisable. Where this is not possible, the user-FTP should make sure to allow for possible clock skew when comparing times.

When attempting to restart a STORe, the User FTP can use the MDTM command to discover the modification time of the partially transferred file. If it is older than the modification time of the file that is about to be STORed, then most likely the source file has changed, and it would be unsafe to restart the file transfer.

Note that using MLST (described below), where available, can provide this information and much more, thus giving an even better indication that a file has changed and that restarting a transfer would not give valid results.

Note that this is applicable to any RESTart attempt, regardless of the mode of the file transfer.

3.1. Syntax

The syntax for the MDTM command is:

```
   mdtm          = "MdTm" SP pathname CRLF
```

As with all FTP commands, the "MDTM" command label is interpreted in a case-insensitive manner.

The "pathname" specifies an object in the NVFS that may be the object of a RETR command. Attempts to query the modification time of files that exist but are unable to be retrieved may generate an error-response, or can result in a positive response carrying a time-val with an unspecified value, the choice being made by the server-PI.

The server-PI will respond to the MDTM command with a 213 reply giving the last modification time of the file whose pathname was supplied, or a 550 reply if the file does not exist, the modification time is unavailable, or some other error has occurred.

```
   mdtm-response = "213" SP time-val CRLF /
                   error-response
```

Note that when the 213 response is issued, that is, when there is no error, the format MUST be exactly as specified. Multi-line responses are not permitted.

3.2. Error Responses

Where the command is correctly parsed but the modification time is not available, either because the pathname identifies no existing entity or because the information is not available for the entity named, then a 550 reply should be sent. Where the command cannot be correctly parsed, a 500 or 501 reply should be sent, as specified in [3]. Various 4xy replies are also possible in appropriate circumstances.

3.3. FEAT Response for MDTM

When replying to the FEAT command [6], a server-FTP process that supports the MDTM command MUST include a line containing the single word "MDTM". This MAY be sent in upper or lower case or a mixture of both (it is case insensitive), but SHOULD be transmitted in upper case only. That is, the response SHOULD be:

```
C> Feat
S> 211- <any descriptive text>
S>  ...
S>  MDTM
S>  ...
S> 211 End
```

The ellipses indicate place holders where other features may be included, but are not required. The one-space indentation of the feature lines is mandatory [6].

3.4. MDTM Examples

If we assume the existence of three files, A B and C, a directory D, two files with names that end with the string "ile6", and no other files at all, then the MDTM command may behave as indicated. The "C>" lines are commands from user-PI to server-PI, the "S>" lines are server-PI replies.

```
C> MDTM A
S> 213 19980615100045.014
C> MDTM B
S> 213 19980615100045.014
C> MDTM C
S> 213 19980705132316
C> MDTM D
S> 550 D is not retrievable
C> MDTM E
S> 550 No file named "E"
C> mdtm file6
S> 213 19990929003355
C> MdTm 19990929043300 File6
S> 213 19991005213102
C> MdTm 19990929043300 file6
S> 550 19990929043300 file6: No such file or directory.
```

From that we can conclude that both A and B were last modified at the same time (to the nearest millisecond), and that C was modified 20 days and several hours later.

The times are in GMT, so file A was modified on the 15th of June, 1998, at approximately 11am in London (summer time was then in effect), or perhaps at 8pm in Melbourne, Australia, or at 6am in New York. All of those represent the same absolute time, of course. The location where the file was modified, and consequently the local wall clock time at that location, is not available.

There is no file named "E" in the current directory, but there are files named both "file6" and "19990929043300 File6". The modification times of those files were obtained. There is no file named "19990929043300 file6".

4. File SIZE

The FTP command, SIZE OF FILE (SIZE), is used to obtain the transfer size of a file from the server-FTP process. This is the exact number of octets (8 bit bytes) that would be transmitted over the data connection should that file be transmitted. This value will change depending on the current STRUcture, MODE, and TYPE of the data connection or of a data connection that would be created were one created now. Thus, the result of the SIZE command is dependent on the currently established STRU, MODE, and TYPE parameters.

The SIZE command returns how many octets would be transferred if the file were to be transferred using the current transfer structure, mode, and type. This command is normally used in conjunction with the RESTART (REST) command when STORing a file to a remote server in STREAM mode, to determine the restart point. The server-PI might need to read the partially transferred file, do any appropriate conversion, and count the number of octets that would be generated when sending the file in order to correctly respond to this command. Estimates of the file transfer size MUST NOT be returned; only precise information is acceptable.

4.1. Syntax

The syntax of the SIZE command is:

```
size          = "Size" SP pathname CRLF
```

The server-PI will respond to the SIZE command with a 213 reply giving the transfer size of the file whose pathname was supplied, or an error response if the file does not exist, the size is unavailable, or some other error has occurred. The value returned is in a format suitable for use with the RESTART (REST) command for mode STREAM, provided the transfer mode and type are not altered.

```
size-response = "213" SP 1*DIGIT CRLF /
                error-response
```

Note that when the 213 response is issued, that is, when there is no error, the format MUST be exactly as specified. Multi-line responses are not permitted.

4.2. Error Responses

Where the command is correctly parsed but the size is not available, perhaps because the pathname identifies no existing entity or because the entity named cannot be transferred in the current MODE and TYPE (or at all), then a 550 reply should be sent. Where the command cannot be correctly parsed, a 500 or 501 reply should be sent, as specified in [3]. The presence of the 550 error response to a SIZE command MUST NOT be taken by the client as an indication that the file cannot be transferred in the current MODE and TYPE. A server may generate this error for other reasons -- for instance if the processing overhead is considered too great. Various 4xy replies are also possible in appropriate circumstances.

4.3. FEAT Response for SIZE

When replying to the FEAT command [6], a server-FTP process that supports the SIZE command MUST include a line containing the single word "SIZE". This word is case insensitive, and MAY be sent in any mixture of upper or lower case, however it SHOULD be sent in upper case. That is, the response SHOULD be:

```
C> FEAT
S> 211- <any descriptive text>
S>  ...
S>  SIZE
S>  ...
S> 211 END
```

The ellipses indicate place holders where other features may be included, and are not required. The one-space indentation of the feature lines is mandatory [6].

4.4. Size Examples

Consider a text file "Example" stored on a Unix(TM) server where each end of line is represented by a single octet. Assume the file contains 112 lines, and 1830 octets total. Then the SIZE command would produce:

```
C> TYPE I
S> 200 Type set to I.
C> size Example
S> 213 1830
C> TYPE A
S> 200 Type set to A.
C> Size Example
S> 213 1942
```

Notice that with TYPE=A the SIZE command reports an extra 112 octets. Those are the extra octets that need to be inserted, one at the end of each line, to provide correct end-of-line semantics for a transfer using TYPE=A. Other systems might need to make other changes to the transfer format of files when converting between TYPEs and MODEs. The SIZE command takes all of that into account.

Since calculating the size of a file with this degree of precision may take considerable effort on the part of the server-PI, user-PIs should not used this command unless this precision is essential (such as when about to restart an interrupted transfer). For other uses, the "Size" fact of the MLST command (see section 7.5.7) ought be requested.

5. Restart of Interrupted Transfer (REST)

To avoid having to resend the entire file if the file is only partially transferred, both sides need some way to agree on where in the data stream to restart the data transfer.

The FTP specification [3] includes three modes of data transfer, STREAM, Block, and Compressed. In Block and Compressed modes, the data stream that is transferred over the data connection is formatted, allowing the embedding of restart markers into the stream. The sending DTP can include a restart marker with whatever information it needs to be able to restart a file transfer at that point. The receiving DTP can keep a list of these restart markers, and correlate them with how the file is being saved. To restart the file transfer, the receiver just sends back that last restart marker, and both sides know how to resume the data transfer. Note that there are some flaws in the description of the restart mechanism in STD 9, RFC 959 [3]. See section 4.1.3.4 of RFC 1123 [9] for the corrections.

5.1. Restarting in STREAM Mode

In STREAM mode, the data connection contains just a stream of unformatted octets of data. Explicit restart markers thus cannot be inserted into the data stream, they would be indistinguishable from data. For this reason, the FTP specification [3] did not provide the ability to do restarts in stream mode. However, there is not really a need to have explicit restart markers in this case, as restart markers can be implied by the octet offset into the data stream.

Because the data stream defines the file in STREAM mode, a different data stream would represent a different file. Thus, an offset will always represent the same position within a file. On the other hand, in other modes than STREAM, the same file can be transferred using quite different octet sequences and yet be reconstructed into the one identical file. Thus an offset into the data stream in transfer modes other than STREAM would not give an unambiguous restart point.

If the data representation TYPE is IMAGE and the STRUcture is File, for many systems the file will be stored exactly in the same format as it is sent across the data connection. It is then usually very easy for the receiver to determine how much data was previously received, and notify the sender of the offset where the transfer should be restarted. In other representation types and structures more effort will be required, but it remains always possible to determine the offset with finite, but perhaps non-negligible, effort. In the worst case, an FTP process may need to open a data connection to itself, set the appropriate transfer type and structure, and actually transmit the file, counting the transmitted octets.

If the user-FTP process is intending to restart a retrieve, it will directly calculate the restart marker and send that information in the RESTart command. However, if the user-FTP process is intending to restart sending the file, it needs to be able to determine how much data was previously sent, and correctly received and saved. A new FTP command is needed to get this information. This is the purpose of the SIZE command, as documented in section 4.

5.2. Error Recovery and Restart

STREAM mode transfers with FILE STRUcture may be restarted even though no restart marker has been transferred in addition to the data itself. This is done by using the SIZE command, if needed, in combination with the RESTART (REST) command, and one of the standard file transfer commands.

When using TYPE ASCII or IMAGE, the SIZE command will return the number of octets that would actually be transferred if the file were

to be sent between the two systems, i.e., with type IMAGE, the SIZE normally would be the number of octets in the file. With type ASCII, the SIZE would be the number of octets in the file including any modifications required to satisfy the TYPE ASCII CR-LF end-of-line convention.

5.3. Syntax

The syntax for the REST command when the current transfer mode is STREAM is:

```
rest          = "Rest" SP 1*DIGIT CRLF
```

The numeric value gives the number of octets of the immediately-following transfer to not actually send, effectively causing the transmission to be restarted at a later point. A value of zero effectively disables restart, causing the entire file to be transmitted. The server-PI will respond to the REST command with a 350 reply, indicating that the REST parameter has been saved, and that another command, which should be either RETR or STOR, should then follow to complete the restart.

```
rest-response = "350" SP *TCHAR CRLF /
                error-response
```

Server-FTP processes may permit transfer commands other than RETR and STOR, such as APPE and STOU, to complete a restart; however, this is not recommended. STOU (store unique) is undefined in this usage, as storing the remainder of a file into a unique file name is rarely going to be useful. If APPE (append) is permitted, it MUST act identically to STOR when a restart marker has been set. That is, in both cases, octets from the data connection are placed into the file at the location indicated by the restart marker value.

The REST command is intended to complete a failed transfer. Use with RETR is comparatively well defined in all cases, as the client bears the responsibility of merging the retrieved data with the partially retrieved file. It may choose to use the data obtained other than to complete an earlier transfer, or to re-retrieve data that had been retrieved before. With STOR, however, the server must insert the data into the file named. The results are undefined if a client uses REST to do other than restart to complete a transfer of a file that had previously failed to completely transfer. In particular, if the restart marker set with a REST command is not at the end of the data currently stored at the server, as reported by the server, or if insufficient data are provided in a STOR that follows a REST to extend the destination file to at least its previous size, then the effects are undefined.

The REST command must be the last command issued before the data transfer command that is to cause a restarted, rather than a complete, file transfer. The effect of issuing a REST command at any other time is undefined. The server-PI may react to a badly positioned REST command by issuing an error response to the following command, not being a restartable data transfer command, or it may save the restart value and apply it to the next data transfer command, or it may silently ignore the inappropriate restart attempt. Because of this, a user-PI that has issued a REST command, but that has not successfully transmitted the following data transfer command for any reason, should send another REST command before the next data transfer command. If that transfer is not to be restarted, then "REST 0" should be issued.

An error response will follow a REST command only when the server does not implement the command, or when the restart marker value is syntactically invalid for the current transfer mode (e.g., in STREAM mode, something other than one or more digits appears in the parameter to the REST command). Any other errors, including such problems as restart marker out of range, should be reported when the following transfer command is issued. Such errors will cause that transfer request to be rejected with an error indicating the invalid restart attempt.

5.4. FEAT Response for REST

Where a server-FTP process supports RESTart in STREAM mode, as specified here, it MUST include, in the response to the FEAT command [6], a line containing exactly the string "REST STREAM". This string is not case sensitive, but it SHOULD be transmitted in upper case. Where REST is not supported at all or supported only in block or compressed modes, the REST line MUST NOT be included in the FEAT response. Where required, the response SHOULD be:

```
C> feat
S> 211- <any descriptive text>
S>  ...
S>  REST STREAM
S>  ...
S> 211 end
```

The ellipses indicate place holders where other features may be included, and are not required. The one-space indentation of the feature lines is mandatory [6].

5.5. REST Example

Assume that the transfer of a largish file has previously been interrupted after 802816 octets had been received, that the previous transfer was with TYPE=I, and that it has been verified that the file on the server has not since changed.

```
C> TYPE I
S> 200 Type set to I.
C> PORT 127,0,0,1,15,107
S> 200 PORT command successful.
C> REST 802816
S> 350 Restarting at 802816. Send STORE or RETRIEVE
C> RETR cap60.pl198.tar
S> 150 Opening BINARY mode data connection
[...]
S> 226 Transfer complete.
```

6. A Trivial Virtual File Store (TVFS)

Traditionally, FTP has placed almost no constraints upon the file store (NVFS) provided by a server. This specification does not alter that. However, it has become common for servers to attempt to provide at least file system naming conventions modeled loosely upon those of the UNIX(TM) file system. This is a tree-structured file system, built of directories, each of which can contain other directories, or other kinds of files, or both. Each file and directory has a name relative to the directory that contains it, except for the directory at the root of the tree, which is contained in no other directory, and hence has no name of its own.

That which has so far been described is perfectly consistent with the standard FTP NVFS and access mechanisms. The "CWD" command is used to move from one directory to an embedded directory. "CDUP" may be provided to return to the parent directory, and the various file manipulation commands ("RETR", "STOR", the rename commands, etc.) are used to manipulate files within the current directory.

However, it is often useful to be able to reference files other than by changing directories, especially as FTP provides no guaranteed mechanism to return to a previous directory. The Trivial Virtual File Store (TVFS), if implemented, provides that mechanism.

6.1. TVFS File Names

Where a server implements the TVFS, no elementary file name shall contain the character "/". Where the underlying natural file store permits files, or directories, to contain the "/" character in their names, a server-PI implementing TVFS must encode that character in some manner whenever file or directory names are being returned to the user-PI, and reverse that encoding whenever such names are being accepted from the user-PI.

The encoding method to be used is not specified here. Where some other character is illegal in file and directory names in the underlying file store, a simple transliteration may be sufficient. Where there is no suitable substitute character a more complex encoding scheme, possibly using an escape character, is likely to be required.

With the one exception of the unnamed root directory, a TVFS file name may not be empty. That is, all other file names contain at least one character.

With the sole exception of the "/" character, any valid IS10646 character [10] may be used in a TVFS file name. When transmitted, file name characters are encoded using the UTF-8 encoding [2]. Note that the two-character sequence CR LF occurring in a file name will make that name impossible to transmit over a data connection. Consequently, it should be avoided, or if that is impossible to achieve, it MUST be encoded in some reversible way.

6.2. TVFS Pathnames

A TVFS "Pathname" combines the file or directory name of a target file or directory, with the directory names of zero or more enclosing directories, so as to allow the target file or directory to be referenced other than when the server’s "current working directory" is the directory directly containing the target file or directory.

By definition, every TVFS file or directory name is also a TVFS pathname. Such a pathname is valid to reference the file from the directory containing the name, that is, when that directory is the server-FTP’s current working directory.

Other TVFS pathnames are constructed by prefixing a pathname by a name of a directory from which the path is valid, and separating the two with the "/" character. Such a pathname is valid to reference the file or directory from the directory containing the newly added directory name.

Where a pathname has been extended to the point where the directory added is the unnamed root directory, the pathname will begin with the "/" character. Such a path is known as a fully qualified pathname. Fully qualified paths may, obviously, not be further extended, as, by definition, no directory contains the root directory. Being unnamed, it cannot be represented in any other directory. A fully qualified pathname is valid to reference the named file or directory from any location (that is, regardless of what the current working directory may be) in the virtual file store.

Any pathname that is not a fully qualified pathname may be referred to as a "relative pathname" and will only correctly reference the intended file when the current working directory of the server-FTP is a directory from which the relative pathname is valid.

As a special case, the pathname "/" is defined to be a fully qualified pathname referring to the root directory. That is, the root directory does not have a directory (or file) name, but does have a pathname. This special pathname may be used only as is as a reference to the root directory. It may not be combined with other pathnames using the rules above, as doing so would lead to a pathname containing two consecutive "/" characters, which is an undefined sequence.

6.2.1. Notes

+ It is not required, or expected, that there be only one fully qualified pathname that will reference any particular file or directory.

+ As a caveat, though the TVFS file store is basically tree structured, there is no requirement that any file or directory have only one parent directory.

+ As defined, no TVFS pathname will ever contain two consecutive "/" characters. Such a name is not illegal however, and may be defined by the server for any purpose that suits it. Clients implementing this specification should not assume any semantics for such names.

+ Similarly, other than the special case path that refers to the root directory, no TVFS pathname constructed as defined here will ever end with the "/" character. Such names are also not illegal, but are undefined.

+ While any legal IS10646 character is permitted to occur in a TVFS file or directory name, other than "/", server FTP implementations are not required to support all possible IS10646 characters. The

subset supported is entirely at the discretion of the server. The case (where it exists) of the characters that make up file, directory, and pathnames may be significant. Unless determined otherwise by means unspecified here, clients should assume that all such names are comprised of characters whose case is significant. Servers are free to treat case (or any other attribute) of a name as irrelevant, and hence map two names that appear to be distinct onto the same underlying file.

+ There are no defined "magic" names, like ".", ".." or "C:". Servers may implement such names, with any semantics they choose, but are not required to do so.

+ TVFS imposes no particular semantics or properties upon files, guarantees no access control schemes, or any of the other common properties of a file store. Only the naming scheme is defined.

6.3. FEAT Response for TVFS

In response to the FEAT command [6] a server that wishes to indicate support for the TVFS as defined here will include a line that begins with the four characters "TVFS" (in any case, or mixture of cases, upper case is not required). Servers SHOULD send upper case.

Such a response to the FEAT command MUST NOT be returned unless the server implements TVFS as defined here.

Later specifications may add to the TVFS definition. Such additions should be notified by means of additional text appended to the TVFS feature line. Such specifications, if any, will define the extra text.

Until such a specification is defined, servers should not include anything after "TVFS" in the TVFS feature line. Clients, however, should be prepared to deal with arbitrary text following the four defined characters, and simply ignore it if unrecognized.

A typical response to the FEAT command issued by a server implementing only this specification would be:

```
C> feat
S> 211- <any descriptive text>
S>  ...
S>  TVFS
S>  ...
S> 211 end
```

The ellipses indicate place holders where other features may be included, but are not required. The one-space indentation of the feature lines is mandatory [6] and is not counted as one of the first four characters for the purposes of this feature listing.

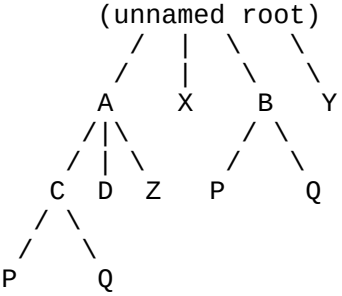
The TVFS feature adds no new commands to the FTP command repertoire.

6.4. OPTS for TVFS

There are no options in this TVFS specification, and hence there is no OPTS command defined.

6.5. TVFS Examples

Assume a TVFS file store is comprised of a root directory, which contains two directories (A and B) and two non-directory files (X and Y). The A directory contains two directories (C and D) and one other file (Z). The B directory contains just two non-directory files (P and Q) and the C directory also two non-directory files (also named P and Q, by chance). The D directory is empty, that is, contains no files or directories. This structure may depicted graphically as...



Given this structure, the following fully qualified pathnames exist.

```
/
/A
/B
/X
/Y
/A/C
/A/D
/A/Z
/A/C/P
/A/C/Q
/B/P
/B/Q
```

It is clear that none of the paths / /A /B or /A/D refer to the same directory, as the contents of each is different. Nor do any of / /A /A/C or /A/D. However /A/C and /B might be the same directory, there is insufficient information given to tell. Any of the other pathnames (/X /Y /A/Z /A/C/P /A/C/Q /B/P and /B/Q) may refer to the same underlying files, in almost any combination.

If the current working directory of the server-FTP is /A then the following pathnames, in addition to all the fully qualified pathnames, are valid

C
D
Z
C/P
C/Q

These all refer to the same files or directories as the corresponding fully qualified path with "/A/" prepended.

That those pathnames all exist does not imply that the TVFS sever will necessarily grant any kind of access rights to the named paths, or that access to the same file via different pathnames will necessarily be granted equal rights.

None of the following relative paths are valid when the current directory is /A

A
B
X
Y
B/P
B/Q
P
Q

Any of those could be made valid by changing the server-FTP's current working directory to the appropriate directory. Note that the paths "P" and "Q" might refer to different files depending upon which directory is selected to cause those to become valid TVFS relative paths.

7. Listings for Machine Processing (MLST and MLSD)

The MLST and MLSD commands are intended to standardize the file and directory information returned by the server-FTP process. These commands differ from the LIST command in that the format of the replies is strictly defined although extensible.

Two commands are defined, MLST and MLSD. MLST provides data about exactly the object named on its command line, and no others. MLSD, on the other, lists the contents of a directory if a directory is named, otherwise a 501 reply is returned. In either case, if no object is named, the current directory is assumed. That will cause MLST to send a one-line response, describing the current directory itself, and MLSD to list the contents of the current directory.

In the following, the term MLSx will be used wherever either MLST or MLSD may be inserted.

The MLST and MLSD commands also extend the FTP protocol as presented in STD 9, RFC 959 [3] and STD 3, RFC 1123 [9] to allow that transmission of 8-bit data over the control connection. Note this is not specifying character sets which are 8-bit, but specifying that FTP implementations are to specifically allow the transmission and reception of 8-bit bytes, with all bits significant, over the control connection. That is, all 256 possible octet values are permitted. The MLSx command allows both UTF-8/Unicode and "raw" forms as arguments, and in responses both to the MLST and MLSD commands, and all other FTP commands which take pathnames as arguments.

7.1. Format of MLSx Requests

The MLST and MLSD commands each allow a single optional argument. This argument may be either a directory name or, for MLST only, a file name. For these purposes, a "file name" is the name of any entity in the server NVFS which is not a directory. Where TVFS is supported, any TVFS relative pathname valid in the current working directory, or any TVFS fully qualified pathname, may be given. If a directory name is given then MLSD must return a listing of the contents of the named directory, otherwise it issues a 501 reply, and does not open a data connection. In all cases for MLST, a single set of fact lines (usually a single fact line) containing the information about the named file or directory shall be returned over the control connection, without opening a data connection.

If no argument is given then MLSD must return a listing of the contents of the current working directory, and MLST must return a listing giving information about the current working directory itself. For these purposes, the contents of a directory are whatever

file or directory names (not pathnames) the server-PI will allow to be referenced when the current working directory is the directory named, and which the server-PI desires to reveal to the user-PI. Note that omitting the argument is the only defined way to obtain a listing of the current directory, unless a pathname that represents the directory happens to be known. In particular, there is no defined shorthand name for the current directory. This does not prohibit any particular server-PI implementing such a shorthand.

No title, header, or summary, lines, or any other formatting, other than as is specified below, is ever returned in the output of an MLST or MLSD command.

If the Client-FTP sends an invalid argument, the server-FTP MUST reply with an error code of 501.

The syntax for the MLSx command is:

```
mlst             = "MLst" [ SP pathname ] CRLF
mlsd             = "MLsD" [ SP pathname ] CRLF
```

7.2. Format of MLSx Response

The format of a response to an MLSx command is as follows:

```
mlst-response    = control-response / error-response
mlsd-response    = ( initial-response final-response ) /
                   error-response

control-response = "250-" [ response-message ] CRLF
                   1*( SP entry CRLF )
                   "250" [ SP response-message ] CRLF

initial-response = "150" [ SP response-message ] CRLF
final-response   = "226" SP response-message CRLF

response-message = *TCHAR

data-response    = *( entry CRLF )

entry            = [ facts ] SP pathname
facts            = 1*( fact ";" )
fact             = factname "=" value
factname         = "Size" / "Modify" / "Create" /
                   "Type" / "Unique" / "Perm" /
                   "Lang" / "Media-Type" / "CharSet" /
                   os-depend-fact / local-fact
os-depend-fact   = <IANA assigned OS name> "." token
```

```
local-fact       = "X." token
value            = *SCHAR
```

Upon receipt of an MLSx command, the server will verify the parameter, and if invalid return an error-response. For this purpose, the parameter should be considered to be invalid if the client issuing the command does not have permission to perform the requested operation.

If the parameter is valid, then for an MLST command, the server-PI will send the first (leading) line of the control response, the entry for the pathname given, or the current directory if no pathname was provided, and the terminating line. Normally exactly one entry would be returned, more entries are permitted only when required to represent a file that is to have multiple "Type" facts returned. In this case, the pathname component of every response MUST be identical.

Note that for MLST the fact set is preceded by a space. That is provided to guarantee that the fact set cannot be accidentally interpreted as the terminating line of the control response, but is required even when that would not be possible. Exactly one space exists between the set of facts and the pathname. Where no facts are present, there will be exactly two leading spaces before the pathname. No spaces are permitted in the facts, any other spaces in the response are to be treated as being a part of the pathname.

If the command was an MLSD command, the server will open a data connection as indicated in section 3.2 of STD 9, RFC 959 [3]. If that fails, the server will return an error-response. If all is OK, the server will return the initial-response, send the appropriate data-response over the new data connection, close that connection, and then send the final-response over the control connection. The grammar above defines the format for the data-response, which defines the format of the data returned over the data connection established.

The data connection opened for a MLSD response shall be a connection as if the "TYPE L 8", "MODE S", and "STRU F" commands had been given, whatever FTP transfer type, mode and structure had actually been set, and without causing those settings to be altered for future commands. That is, this transfer type shall be set for the duration of the data connection established for this command only. While the content of the data sent can be viewed as a series of lines, implementations should note that there is no maximum line length defined. Implementations should be prepared to deal with arbitrarily long lines.

The facts part of the specification would contain a series of "file facts" about the file or directory named on the same line. Typical information to be presented would include file size, last modification time, creation time, a unique identifier, and a file/directory flag.

The complete format for a successful reply to the MLSD command would be:

```
facts SP pathname CRLF
facts SP pathname CRLF
facts SP pathname CRLF
...
```

Note that the format is intended for machine processing, not human viewing, and as such the format is very rigid. Implementations MUST NOT vary the format by, for example, inserting extra spaces for readability, replacing spaces by tabs, including header or title lines, or inserting blank lines, or in any other way alter this format. Exactly one space is always required after the set of facts (which may be empty). More spaces may be present on a line if, and only if, the pathname presented contains significant spaces. The set of facts must not contain any spaces anywhere inside it. Facts should be provided in each output line only if they both provide relevant information about the file named on the same line, and they are in the set requested by the user-PI. See section 7.9 (page 51). There is no requirement that the same set of facts be provided for each file, or that the facts presented occur in the same order for each file.

7.2.1. Error Responses to MLSx commands

Many of the 4xy and 5xy responses defined in section 4.2 of STD 9, RFC 959 [3] are possible in response to the MLST and MLSD commands. In particular, syntax errors can generate 500 or 501 replies. Giving a pathname that exists but is not a directory as the argument to a MLSD command generates a 501 reply. Giving a name that does not exist, or for which access permission (to obtain directory information as requested) is not granted will elicit a 550 reply. Other replies (530, 553, 503, 504, and any of the 4xy replies) are also possible in appropriate circumstances.

7.3. File Name Encoding

An FTP implementation supporting the MLSx commands must be 8-bit clean. This is necessary in order to transmit UTF-8 encoded file names. This specification recommends the use of UTF-8 encoded file

names. FTP implementations SHOULD use UTF-8 whenever possible to encourage the maximum inter-operability.

File names are not restricted to UTF-8, however treatment of arbitrary character encodings is not specified by this standard. Applications are encouraged to treat non-UTF-8 encodings of file names as octet sequences.

Note that this encoding is unrelated to that of the contents of the file, even if the file contains character data.

Further information about file name encoding for FTP may be found in "Internationalization of the File Transfer Protocol" [7].

7.3.1. Notes about the File Name

The file name returned in the MLST response should be the same name as was specified in the MLST command, or, where TVFS is supported, a fully qualified TVFS path naming the same file. Where no argument was given to the MLST command, the server-PI may either include an empty file name in the response, or it may supply a name that refers to the current directory, if such a name is available. Where TVFS is supported, a fully qualified pathname of the current directory SHOULD be returned.

File names returned in the output from an MLSD command SHOULD be unqualified names within the directory named, or the current directory if no argument was given. That is, the directory named in the MLSD command SHOULD NOT appear as a component of the file names returned.

If the server-FTP process is able, and the "type" fact is being returned, it MAY return in the MLSD response, an entry whose type is "cdir", which names the directory from which the contents of the listing were obtained. Where TVFS is supported, the name MAY be the fully qualified pathname of the directory, or MAY be any other pathname that is valid to refer to that directory from the current working directory of the server-FTP. Where more than one name exists, multiple of these entries may be returned. In a sense, the "cdir" entry can be viewed as a heading for the MLSD output. However, it is not required to be the first entry returned, and may occur anywhere within the listing.

When TVFS is supported, a user-PI can refer to any file or directory in the listing by combining a type "cdir" name, with the appropriate name from the directory listing using the procedure defined in section 6.2.

Alternatively, whether TVFS is supported or not, the user-PI can issue a CWD command ([3]) giving a name of type "cdir" from the listing returned, and from that point reference the files returned in the MLSD response from which the cdir was obtained by using the file name components of the listing.

7.4. Format of Facts

The "facts" for a file in a reply to a MLSx command consist of information about that file. The facts are a series of keyword=value pairs each followed by semi-colon (";") characters. An individual fact may not contain a semi-colon in its name or value. The complete series of facts may not contain the space character. See the definition or "RCHAR" in section 2.1 for a list of the characters that can occur in a fact value. Not all are applicable to all facts.

A sample of a typical series of facts would be: (spread over two lines for presentation here only)

```
size=4161;lang=en-US;modify=19970214165800;create=19961001124534;
type=file;x.myfact=foo,bar;
```

7.5. Standard Facts

This document defines a standard set of facts as follows:

```
size       -- Size in octets
modify     -- Last modification time
create     -- Creation time
type       -- Entry type
unique     -- Unique id of file/directory
perm       -- File permissions, whether read, write, execute is
              allowed for the login id.
lang       -- Language of the file name per IANA [11] registry.
media-type -- MIME media-type of file contents per IANA registry.
charset    -- Character set per IANA registry (if not UTF-8)
```

Fact names are case-insensitive. Size, size, SIZE, and SiZe are the same fact.

Further operating system specific keywords could be specified by using the IANA operating system name as a prefix (examples only):

```
OS/2.ea   -- OS/2 extended attributes
MACOS.rf  -- MacIntosh resource forks
UNIX.mode -- Unix file modes (permissions)
```

Implementations may define keywords for experimental, or private use. All such keywords MUST begin with the two character sequence "x.". As type names are case independent, "x." and "X." are equivalent. For example:

```
x.ver  -- Version information
x.desc -- File description
x.type -- File type
```

7.5.1. The Type Fact

The type fact needs a special description. Part of the problem with current practices is deciding when a file is a directory. If it is a directory, is it the current directory, a regular directory, or a parent directory? The MLST specification makes this unambiguous using the type fact. The type fact given specifies information about the object listed on the same line of the MLST response.

Five values are possible for the type fact:

```
file         -- a file entry
cdir         -- the listed directory
pdir         -- a parent directory
dir          -- a directory or sub-directory
OS.name=type -- an OS or file system dependent file type
```

The syntax is defined to be:

```
type-fact       = type-label "=" type-val
type-label      = "Type"
type-val        = "File" / "cdir" / "pdir" / "dir" /
                  os-type
```

The value of the type fact (the "type-val") is a case independent string.

7.5.1.1. type=file

The presence of the type=file fact indicates the listed entry is a file containing non-system data. That is, it may be transferred from one system to another of quite different characteristics, and perhaps still be meaningful.

7.5.1.2. type=cdir

The type=cdir fact indicates the listed entry contains a pathname of the directory whose contents are listed. An entry of this type will only be returned as a part of the result of an MLSD command when the

type fact is included, and provides a name for the listed directory, and facts about that directory. In a sense, it can be viewed as representing the title of the listing, in a machine friendly format. It may appear at any point of the listing, it is not restricted to appearing at the start, though frequently may do so, and may occur multiple times. It MUST NOT be included if the type fact is not included, or there would be no way for the user-PI to distinguish the name of the directory from an entry in the directory.

Where TVFS is supported by the server-FTP, this name may be used to construct pathnames with which to refer to the files and directories returned in the same MLSD output (see section 6.2). These pathnames are only expected to work when the server-PI's position in the NVFS file tree is the same as its position when the MLSD command was issued, unless a fully qualified pathname results.

Where TVFS is not supported, the only defined semantics associated with a "type=cdir" entry are that, provided the current working directory of the server-PI has not been changed, a pathname of type "cdir" may be used as an argument to a CWD command, which will cause the current directory of the server-PI to change so that the directory that was listed in its current working directory.

7.5.1.3. type=dir

If present, the type=dir entry gives the name of a directory. Such an entry typically cannot be transferred from one system to another using RETR, etc., but should (permissions permitting) be able to be the object of an MLSD command.

7.5.1.4. type=pdir

If present, which will occur only in the response to a MLSD command when the type fact is included, the type=pdir entry represents a pathname of the parent directory of the listed directory. As well as having the properties of a type=dir, a CWD command that uses the pathname from this entry should change the user to a parent directory of the listed directory. If the listed directory is the current directory, a CDUP command may also have the effect of changing to the named directory. User-FTP processes should note not all responses will include this information, and that some systems may provide multiple type=pdir responses.

Where TVFS is supported, a "type=pdir" name may be a relative pathname, or a fully qualified pathname. A relative pathname will be relative to the directory being listed, not to the current directory of the server-PI at the time.

For the purposes of this type value, a "parent directory" is any directory in which there is an entry of type=dir that refers to the directory in which the type=pdir entity was found. Thus it is not required that all entities with type=pdir refer to the same directory. The "unique" fact (if supported and supplied) can be used to determine whether there is a relationship between the type=pdir entries or not.

7.5.1.5. System Defined Types

Files types that are specific to a specific operating system, or file system, can be encoded using the "OS." type names. The format is:

```
os-type   = "OS." os-name "=" os-kind
os-name   = <an IANA registered operating system name>
os-kind   = token
```

The "os-name" indicates the specific system type that supports the particular localtype. OS specific types are registered by the IANA using the procedures specified in section 10. The "os-kind" provides the system dependent information as to the type of the file listed. The os-name and os-kind strings in an os-type are case independent. "OS.unix=block" and "OS.Unix=BLOCK" represent the same type (or would, if such a type were registered.)

Note: Where the underlying system supports a file type that is essentially an indirect pointer to another file, the NVFS representation of that type should normally be to represent the file that the reference indicates. That is, the underlying basic file will appear more than once in the NVFS, each time with the "unique" fact (see immediately following section) containing the same value, indicating that the same file is represented by all such names. User-PIs transferring the file need then transfer it only once, and then insert their own form of indirect reference to construct alternate names where desired, or perhaps even copy the local file if that is the only way to provide two names with the same content. A file which would be a reference to another file, if only the other file actually existed, may be represented in any OS dependent manner appropriate, or not represented at all.

7.5.1.6. Multiple Types

Where a file is such that it may validly, and sensibly, treated by the server-PI as being of more than one of the above types, then multiple entries should be returned, each with its own "Type" fact of the appropriate type, and each containing the same pathname. This may occur, for example, with a structured file, which may contain sub-files, and where the server-PI permits the structured file to be

treated as a unit, or treated as a directory allowing the sub-files within it to be referenced. When this is done, the pathname returned with each entry MUST be identical to the others representing the same file.

7.5.2. The unique Fact

The unique fact is used to present a unique identifier for a file or directory in the NVFS accessed via a server-FTP process. The value of this fact should be the same for any number of pathnames that refer to the same underlying file. The fact should have different values for names that reference distinct files. The mapping between files, and unique fact tokens should be maintained, and remain consistent, for at least the lifetime of the control connection from user-PI to server-PI.

```
unique-fact  = "Unique" "=" token
```

This fact would be expected to be used by server-FTPs whose host system allows things such as symbolic links so that the same file may be represented in more than one directory on the server. The only conclusion that should be drawn is that if two different names each have the same value for the unique fact, they refer to the same underlying object. The value of the unique fact (the token) should be considered an opaque string for comparison purposes, and is a case dependent value. The tokens "A" and "a" do not represent the same underlying object.

7.5.3. The modify Fact

The modify fact is used to determine the last time the content of the file (or directory) indicated was modified. Any change of substance to the file should cause this value to alter. That is, if a change is made to a file such that the results of a RETR command would differ, then the value of the modify fact should alter. User-PIs should not assume that a different modify fact value indicates that the file contents are necessarily different than when last retrieved. Some systems may alter the value of the modify fact for other reasons, though this is discouraged wherever possible. Also a file may alter, and then be returned to its previous content, which would often be indicated as two incremental alterations to the value of the modify fact.

For directories, this value should alter whenever a change occurs to the directory such that different file names would (or might) be included in MLSD output of that directory.

```
modify-fact  = "Modify" "=" time-val
```

7.5.4. The create Fact

The create fact indicates when a file, or directory, was first created. Exactly what "creation" is for this purpose is not specified here, and may vary from server to server. About all that can be said about the value returned is that it can never indicate a later time than the modify fact.

```
create-fact  = "Create" "=" time-val
```

Implementation Note: Implementors of this fact on UNIX(TM) systems should note that the unix "stat" "st_ctime" field does not give creation time, and that unix file systems do not record creation time at all. Unix (and POSIX) implementations will normally not include this fact.

7.5.5. The perm Fact

The perm fact is used to indicate access rights the current FTP user has over the object listed. Its value is always an unordered sequence of alphabetic characters.

```
perm-fact    = "Perm" "=" *pvals
pvals        = "a" / "c" / "d" / "e" / "f" /
               "l" / "m" / "p" / "r" / "w"
```

There are ten permission indicators currently defined. Many are meaningful only when used with a particular type of object. The indicators are case independent, "d" and "D" are the same indicator.

The "a" permission applies to objects of type=file, and indicates that the APPE (append) command may be applied to the file named.

The "c" permission applies to objects of type=dir (and type=pdir, type=cdir). It indicates that files may be created in the directory named. That is, that a STOU command is likely to succeed, and that STOR and APPE commands might succeed if the file named did not previously exist, but is to be created in the directory object that has the "c" permission. It also indicates that the RNTO command is likely to succeed for names in the directory.

The "d" permission applies to all types. It indicates that the object named may be deleted, that is, that the RMD command may be applied to it if it is a directory, and otherwise that the DELE command may be applied to it.

The "e" permission applies to the directory types. When set on an object of type=dir, type=cdir, or type=pdir it indicates that a CWD

command naming the object should succeed, and the user should be able to enter the directory named. For type=pdir it also indicates that the CDUP command may succeed (if this particular pathname is the one to which a CDUP would apply.)

The "f" permission for objects indicates that the object named may be renamed - that is, may be the object of an RNFR command.

The "l" permission applies to the directory file types, and indicates that the listing commands, LIST, NLST, and MLSD may be applied to the directory in question.

The "m" permission applies to directory types, and indicates that the MKD command may be used to create a new directory within the directory under consideration.

The "p" permission applies to directory types, and indicates that objects in the directory may be deleted, or (stretching naming a little) that the directory may be purged. Note: it does not indicate that the RMD command may be used to remove the directory named itself, the "d" permission indicator indicates that.

The "r" permission applies to type=file objects, and for some systems, perhaps to other types of objects, and indicates that the RETR command may be applied to that object.

The "w" permission applies to type=file objects, and for some systems, perhaps to other types of objects, and indicates that the STOR command may be applied to the object named.

Note: That a permission indicator is set can never imply that the appropriate command is guaranteed to work -- just that it might. Other system specific limitations, such as limitations on available space for storing files, may cause an operation to fail, where the permission flags may have indicated that it was likely to succeed. The permissions are a guide only.

Implementation note: The permissions are described here as they apply to FTP commands. They may not map easily into particular permissions available on the server's operating system. Servers are expected to synthesize these permission bits from the permission information available from operating system. For example, to correctly determine whether the "D" permission bit should be set on a directory for a server running on the UNIX(TM) operating system, the server should check that the directory named is empty, and that the user has write permission on both the directory under consideration, and its parent directory.

Some systems may have more specific permissions than those listed here, such systems should map those to the flags defined as best they are able. Other systems may have only more broad access controls. They will generally have just a few possible permutations of permission flags, however they should attempt to correctly represent what is permitted.

7.5.6. The lang Fact

The lang fact describes the natural language of the file name for use in display purposes. Values used here should be taken from the language registry of the IANA. See [12] for the syntax, and procedures, related to language tags.

```
lang-fact  = "Lang" "=" token
```

Server-FTP implementations MUST NOT guess language values. Language values must be determined in an unambiguous way such as file system tagging of language or by user configuration. Note that the lang fact provides no information at all about the content of a file, only about the encoding of its name.

7.5.7. The size Fact

The size fact applies to non-directory file types and should always reflect the approximate size of the file. This should be as accurate as the server can make it, without going to extraordinary lengths, such as reading the entire file. The size is expressed in units of octets of data in the file.

Given limitations in some systems, Client-FTP implementations must understand this size may not be precise and may change between the time of a MLST and RETR operation.

Clients that need highly accurate size information for some particular reason should use the SIZE command as defined in section 4. The most common need for this accuracy is likely to be in conjunction with the REST command described in section 5. The size fact, on the other hand, should be used for purposes such as indicating to a human user the approximate size of the file to be transferred, and perhaps to give an idea of expected transfer completion time.

```
size-fact  = "Size" "=" 1*DIGIT
```

7.5.8. The media-type Fact

The media-type fact represents the IANA media type of the file named, and applies only to non-directory types. The list of values used must follow the guidelines set by the IANA registry.

```
media-type  = "Media-Type" "=" <per IANA guidelines>
```

Server-FTP implementations MUST NOT guess media type values. Media type values must be determined in an unambiguous way such as file system tagging of media-type or by user configuration. This fact gives information about the content of the file named. Both the primary media type, and any appropriate subtype should be given, separated by a slash "/" as is traditional.

7.5.9. The charset Fact

The charset fact provides the IANA character set name, or alias, for the encoded pathnames in a MLSx response. The default character set is UTF-8 unless specified otherwise. FTP implementations SHOULD use UTF-8 if possible to encourage maximum inter-operability. The value of this fact applies to the pathname only, and provides no information about the contents of the file.

```
charset-type  = "Charset" "=" token
```

7.5.10. Required Facts

Servers are not required to support any particular set of the available facts. However, servers SHOULD, if conceivably possible, support at least the type, perm, size, unique, and modify facts.

7.6. System Dependent and Local Facts

By using an system dependent fact, or a local fact, a server-PI may communicate to the user-PI information about the file named that is peculiar to the underlying file system.

7.6.1. System Dependent Facts

System dependent fact names are labeled by prefixing a label identifying the specific information returned by the name of the appropriate operating system from the IANA maintained list of operating system names.

The value of an OS dependent fact may be whatever is appropriate to convey the information available. It must be encoded as a "token" as defined in section 2.1 however.

In order to allow reliable inter-operation between users of system dependent facts, the IANA will maintain a registry of system dependent fact names, their syntax, and the interpretation to be given to their values. Registrations of system dependent facts are to be accomplished according to the procedures of section 10.

7.6.2. Local Facts

Implementations may also make available other facts of their own choosing. As the method of interpretation of such information will generally not be widely understood, server-PIs should be aware that clients will typically ignore any local facts provided. As there is no registration of locally defined facts, it is entirely possible that different servers will use the same local fact name to provide vastly different information. Hence user-PIs should be hesitant about making any use of any information in a locally defined fact without some other specific assurance that the particular fact is one that they do comprehend.

Local fact names all begin with the sequence "X.". The rest of the name is a "token" (see section 2.1). The value of a local fact can be anything at all, provided it can be encoded as a "token".

7.7. MLSx Examples

The following examples are all taken from dialogues between existing FTP clients and servers. Because of this, not all possible variations of possible response formats are shown in the examples. This should not be taken as limiting the options of other server implementors. Where the examples show OS dependent information, that is to be treated as being purely for the purposes of demonstration of some possible OS specific information that could be defined. As at the time of the writing of this document, no OS specific facts or file types have been defined, the examples shown here should not be treated as in any way to be preferred over other possible similar definitions. Consult the IANA registries to determine what types and facts have been defined. Finally also beware that as the examples shown are taken from existing implementations, coded before this document was completed, the possibility of variations between the text of this document and the examples exists. In any such case of inconsistency, the example is to be treated as incorrect.

In the examples shown, only relevant commands and responses have been included. This is not to imply that other commands (including authentication, directory modification, PORT or PASV commands, or similar) would not be present in an actual connection, or were not, in fact, actually used in the examples before editing. Note also that the formats shown are those that are transmitted between client

and server, not formats that would normally ever be reported to the user of the client.

7.7.1. Simple MLST

```
C> PWD
S> 257 "/tmp" is current directory.
C> MLst cap60.pl198.tar.gz
S> 250- Listing cap60.pl198.tar.gz
S>  Type=file;Size=1024990;Perm=r; /tmp/cap60.pl198.tar.gz
S> 250 End
```

The client first asked to be told the current directory of the server. This was purely for the purposes of clarity of this example. The client then requested facts about a specific file. The server returned the "250-" first control-response line, followed by a single line of facts about the file, followed by the terminating "250 " line. The text on the control-response line and the terminating line can be anything the server decides to send. Notice that the fact line is indented by a single space. Notice also that there are no spaces in the set of facts returned, until the single space before the file name. The file name returned on the fact line is a fully qualified pathname of the file listed. The facts returned show that the line refers to a file, that file contains approximately 1024990 bytes, though more or less than that may be transferred if the file is retrieved, and a different number may be required to store the file at the client's file store, and the connected user has permission to retrieve the file but not to do anything else particularly interesting.

7.7.2. MLST of a directory

```
C> PWD
S> 257 "/" is current directory.
C> MLst tmp
S> 250- Listing tmp
S>  Type=dir;Modify=19981107085215;Perm=el; /tmp
S> 250 End
```

Again the PWD is just for the purposes of demonstration for the example. The MLST fact line this time shows that the file listed is a directory, that it was last modified at 08:52:15 on the 7th of November, 1998 UTC, and that the user has permission to enter the directory, and to list its contents, but not to modify it in any way. Again, the fully qualified pathname of the directory listed is given.

7.7.3. MLSD of a directory

```
C> MLSD tmp
S> 150 BINARY connection open for MLSD tmp
D> Type=cdir;Modify=19981107085215;Perm=el; tmp
D> Type=cdir;Modify=19981107085215;Perm=el; /tmp
D> Type=pdir;Modify=19990112030508;Perm=el; ..
D> Type=file;Size=25730;Modify=19940728095854;Perm=; capmux.tar.z
D> Type=file;Size=1830;Modify=19940916055648;Perm=r; hatch.c
D> Type=file;Size=25624;Modify=19951003165342;Perm=r; MacIP-02.txt
D> Type=file;Size=2154;Modify=19950501105033;Perm=r; uar.netbsd.patch
D> Type=file;Size=54757;Modify=19951105101754;Perm=r; iptnnladev.1.0.sit.hqx
D> Type=file;Size=226546;Modify=19970515023901;Perm=r; melbcs.tif
D> Type=file;Size=12927;Modify=19961025135602;Perm=r; tardis.1.6.sit.hqx
D> Type=file;Size=17867;Modify=19961025135602;Perm=r; timelord.1.4.sit.hqx
D> Type=file;Size=224907;Modify=19980615100045;Perm=r; uar.1.2.3.sit.hqx
D> Type=file;Size=1024990;Modify=19980130010322;Perm=r; cap60.pl198.tar.gz
S> 226 MLSD completed
```

In this example notice that there is no leading space on the fact lines returned over the data connection. Also notice that two lines of "type=cdir" have been given. These show two alternate names for the directory listed, one a fully qualified pathname, and the other a local name relative to the servers current directory when the MLSD was performed. Note that all other file names in the output are relative to the directory listed, though the server could, if it chose, give a fully qualified pathname for the "type=pdir" line. This server has chosen not to. The other files listed present a fairly boring set of files that are present in the listed directory. Note that there is no particular order in which they are listed. They are not sorted by file name, by size, or by modify time. Note also that the "perm" fact has an empty value for the file "capmux.tar.z" indicating that the connected user has no permissions at all for that file. This server has chosen to present the "cdir" and "pdir" lines before the lines showing the content of the directory, it is not required to do so. The "size" fact does not provide any meaningful information for a directory, so is not included in the fact lines for the directory types shown.

7.7.4. A More Complex Example

```
C> MLst test
S> 250- Listing test
S>  Type=dir;Perm=el;Unique=keVO1+ZF4 test
S> 250 End
C> MLSD test
S> 150 BINARY connection open for MLSD test
D> Type=cdir;Perm=el;Unique=keVO1+ZF4; test
D> Type=pdir;Perm=e;Unique=keVO1+d?3; ..
D> Type=OS.unix=slink:/foobar;Perm=;Unique=keVO1+4G4; foobar
D> Type=OS.unix=chr-13/29;Perm=;Unique=keVO1+5G4; device
D> Type=OS.unix=blk-11/108;Perm=;Unique=keVO1+6G4; block
D> Type=file;Perm=awr;Unique=keVO1+8G4; writable
D> Type=dir;Perm=cpmel;Unique=keVO1+7G4; promiscuous
D> Type=dir;Perm=;Unique=keVO1+1t2; no-exec
D> Type=file;Perm=r;Unique=keVO1+EG4; two words
D> Type=file;Perm=r;Unique=keVO1+IH4;  leading space
D> Type=file;Perm=r;Unique=keVO1+1G4; file1
D> Type=dir;Perm=cpmel;Unique=keVO1+7G4; incoming
D> Type=file;Perm=r;Unique=keVO1+1G4; file2
D> Type=file;Perm=r;Unique=keVO1+1G4; file3
D> Type=file;Perm=r;Unique=keVO1+1G4; file4
S> 226 MLSD completed
C> MLSD test/incoming
S> 150 BINARY connection open for MLSD test/incoming
D> Type=cdir;Perm=cpmel;Unique=keVO1+7G4; test/incoming
D> Type=pdir;Perm=el;Unique=keVO1+ZF4; ..
D> Type=file;Perm=awdrf;Unique=keVO1+EH4; bar
D> Type=file;Perm=awdrf;Unique=keVO1+LH4;
D> Type=file;Perm=rf;Unique=keVO1+1G4; file5
D> Type=file;Perm=rf;Unique=keVO1+1G4; file6
D> Type=dir;Perm=cpmdelf;Unique=keVO1+!s2; empty
S> 226 MLSD completed
```

For the purposes of this example the fact set requested has been modified to delete the "size" and "modify" facts, and add the "unique" fact. First, facts about a file name have been obtained via MLST. Note that no fully qualified pathname was given this time. That was because the server was unable to determine that information. Then having determined that the file name represents a directory, that directory has been listed. That listing also shows no fully qualified pathname, for the same reason, thus has but a single "type=cdir" line. This directory (which was created especially for the purpose) contains several interesting files. There are some with OS dependent file types, several sub-directories, and several ordinary files.

Not much can be said here about the OS dependent file types, as none of the information shown there should be treated as any more than possibilities. It can be seen that the OS type of the server is "unix" though, which is one of the OS types in the IANA registry of Operating System names.

Of the three directories listed, "no-exec" has no permission granted to this user to access at all. From the "Unique" fact values, it can be determined that "promiscuous" and "incoming" in fact represent the same directory. Its permissions show that the connected user has permission to do essentially anything other than to delete the directory. That directory was later listed. It happens that the directory can not be deleted because it is not empty.

Of the normal files listed, two contain spaces in their names. The file called " leading space" actually contains two spaces in its name, one before the "l" and one between the "g" and the "s". The two spaces that separate the facts from the visible part of the pathname make that clear. The file "writable" has the "a" and "w" permission bits set, and consequently the connected user should be able to STOR or APPE to that file.

The other four file names, "file1", "file2", "file3", and "file4" all represent the same underlying file, as can be seen from the values of the "unique" facts of each. It happens that "file1" and "file2" are Unix "hard" links, and that "file3" and "file4" are "soft" or "symbolic" links to the first two. None of that information is available via standard MLST facts, it is sufficient for the purposes of FTP to note that all represent the same file, and that the same data would be fetched no matter which of them was retrieved, and that all would be simultaneously modified were data stored in any.

Finally, the sub-directory "incoming" is listed. Since "promiscuous" is the same directory there would be no point listing it as well. In that directory, the files "file5" and "file6" represent still more names for the "file1" file we have seen before. Notice the entry between that for "bar" and "file5". Though it is not possible to easily represent it in this document, that shows a file with a name comprising exactly three spaces (" "). A client will have no difficulty determining that name from the output presented to it however. The directory "empty" is, as its name implies, empty, though that is not shown here. It can, however, be deleted, as can file "bar" and the file whose name is three spaces. All the files that reside in this directory can be renamed. This is a consequence of the UNIX semantics of the directory that contains them being modifiable.

7.7.5. More Accurate Time Information

```
C> MLst file1
S> 250- Listing file1
S>  Type=file;Modify=19990929003355.237; file1
S> 250 End
```

In this example, the server-FTP is indicating that "file1" was last modified 237 milliseconds after 00:33:55 UTC on the 29th of September, 1999.

7.7.6. A Different Server

```
C> MLST
S> 250-Begin
S>  type=dir;unique=AQkAAAAAAAABCAAA; /
S> 250 End.
C> MLSD
S> 150 Opening ASCII mode data connection for MLS.
D> type=cdir;unique=AQkAAAAAAAABCAAA; /
D> type=dir;unique=AQkAAAAAAAABEAAA; bin
D> type=dir;unique=AQkAAAAAAAABGAAA; etc
D> type=dir;unique=AQkAAAAAAAAB8AwA; halflife
D> type=dir;unique=AQkAAAAAAAABoAAA; incoming
D> type=dir;unique=AQkAAAAAAAABIAAA; lib
D> type=dir;unique=AQkAAAAAAAABWAEA; linux
D> type=dir;unique=AQkAAAAAAAABKAEA; ncftpd
D> type=dir;unique=AQkAAAAAAAABGAEA; outbox
D> type=dir;unique=AQkAAAAAAAABuAAA; quake2
D> type=dir;unique=AQkAAAAAAAABQAEA; winstuff
S> 226 Listing completed.
C> MLSD linux
S> 150 Opening ASCII mode data connection for MLS.
D> type=cdir;unique=AQkAAAAAAAABWAEA; /linux
D> type=pdir;unique=AQkAAAAAAAABCAAA; /
D> type=dir;unique=AQkAAAAAAAABeAEA; firewall
D> type=file;size=12;unique=AQkAAAAAAAACWAEA; helo_world
D> type=dir;unique=AQkAAAAAAAABYAEA; kernel
D> type=dir;unique=AQkAAAAAAAABmAEA; scripts
D> type=dir;unique=AQkAAAAAAAABkAEA; security
S> 226 Listing completed.
C> MLSD linux/kernel
S> 150 Opening ASCII mode data connection for MLS.
D> type=cdir;unique=AQkAAAAAAAABYAEA; /linux/kernel
D> type=pdir;unique=AQkAAAAAAAABWAEA; /linux
D> type=file;size=6704;unique=AQkAAAAAAAADYAEA; k.config
D> type=file;size=7269221;unique=AQkAAAAAAAACYAEA; linux-2.0.36.tar.gz
D> type=file;size=12514594;unique=AQkAAAAAAAAEYAEA; linux-2.1.130.tar.gz
```

```
S> 226 Listing completed.
```

Note that this server returns its "unique" fact value in quite a different format. It also returns fully qualified pathnames for the "pdir" entry.

7.7.7. Some IANA Files

```
C> MLSD
S> 150 BINARY connection open for MLSD .
D> Type=cdir;Modify=19990219183438; /iana/assignments
D> Type=pdir;Modify=19990112030453; ..
D> Type=dir;Modify=19990219073522; media-types
D> Type=dir;Modify=19990112033515; character-set-info
D> Type=dir;Modify=19990112033529; languages
D> Type=file;Size=44242;Modify=19990217230400; character-sets
D> Type=file;Size=1947;Modify=19990209215600; operating-system-names
S> 226 MLSD completed
C> MLSD media-types
S> 150 BINARY connection open for MLSD media-types
D> Type=cdir;Modify=19990219073522; media-types
D> Type=cdir;Modify=19990219073522; /iana/assignments/media-types
D> Type=pdir;Modify=19990219183438; ..
D> Type=dir;Modify=19990112033045; text
D> Type=dir;Modify=19990219183442; image
D> Type=dir;Modify=19990112033216; multipart
D> Type=dir;Modify=19990112033254; video
D> Type=file;Size=30249;Modify=19990218032700; media-types
S> 226 MLSD completed
C> MLSD character-set-info
S> 150 BINARY connection open for MLSD character-set-info
D> Type=cdir;Modify=19990112033515; character-set-info
D> Type=cdir;Modify=19990112033515; /iana/assignments/character-set-info
D> Type=pdir;Modify=19990219183438; ..
D> Type=file;Size=1234;Modify=19980903020400; windows-1251
D> Type=file;Size=4557;Modify=19980922001400; tis-620
D> Type=file;Size=801;Modify=19970324130000; ibm775
D> Type=file;Size=552;Modify=19970320130000; ibm866
D> Type=file;Size=922;Modify=19960505140000; windows-1258
S> 226 MLSD completed
C> MLSD languages
S> 150 BINARY connection open for MLSD languages
D> Type=cdir;Modify=19990112033529; languages
D> Type=cdir;Modify=19990112033529; /iana/assignments/languages
D> Type=pdir;Modify=19990219183438; ..
D> Type=file;Size=2391;Modify=19980309130000; default
D> Type=file;Size=943;Modify=19980309130000; tags
D> Type=file;Size=870;Modify=19971026130000; navajo
```

```
D> Type=file;Size=699;Modify=19950911140000; no-bok
S> 226 MLSD completed
C> PWD
S> 257 "/iana/assignments" is current directory.
```

This example shows some of the IANA maintained files that are relevant for this specification in MLSD format. Note that these listings have been edited by deleting many entries, the actual listings are much longer.

7.7.8. A Stress Test of Case (In)dependence

The following example is intended to make clear some cases where case dependent strings are permitted in the MLSx commands, and where case independent strings are required.

Note first that the "MLSD" command, shown here as "MlsD" is case independent. Clients may issue this command in any case, or combination of cases, they desire. This is the case for all FTP commands.

```
C> MlsD
S> 150 BINARY connection open for MLSD .
D> Type=pdir;Modify=19990929011228;Perm=el;Unique=keVO1+ZF4; ..
D> Type=file;Size=4096;Modify=19990929011440;Perm=r;Unique=keVO1+Bd8; FILE2
D> Type=file;Size=4096;Modify=19990929011440;Perm=r;Unique=keVO1+aG8; file3
D> Type=file;Size=4096;Modify=19990929011440;Perm=r;Unique=keVO1+ag8; FILE3
D> Type=file;Size=4096;Modify=19990929011440;Perm=r;Unique=keVO1+bD8; file1
D> Type=file;Size=4096;Modify=19990929011440;Perm=r;Unique=keVO1+bD8; file2
D> Type=file;Size=4096;Modify=19990929011440;Perm=r;Unique=keVO1+Ag8; File3
D> Type=file;Size=4096;Modify=19990929011440;Perm=r;Unique=keVO1+bD8; File1
D> Type=file;Size=4096;Modify=19990929011440;Perm=r;Unique=keVO1+Bd8; File2
D> Type=file;Size=4096;Modify=19990929011440;Perm=r;Unique=keVO1+bd8; FILE1
S> 226 MLSD completed
```

Next, notice the labels of the facts. These are also case-independent strings; the server-FTP is permitted to return them in any case desired. User-FTP must be prepared to deal with any case, though it may do this by mapping the labels to a common case if desired.

Then, notice that there are nine objects of "type" file returned. In a case-independent NVFS these would represent three different file names, "file1", "file2", and "file3". With a case-dependent NVFS all nine represent different file names. Either is possible, server-FTPs may implement a case dependent or a case independent NVFS. User-FTPs must allow for case dependent selection of files to manipulate on the server.

Lastly, notice that the value of the "unique" fact is case dependent. In the example shown, "file1", "File1", and "file2" all have the same "unique" fact value "keVO1+bD8", and thus all represent the same underlying file. On the other hand, "FILE1" has a different "unique" fact value ("keVO1+bd8") and hence represents a different file. Similarly, "FILE2" and "File2" are two names for the same underlying file, whereas "file3", "File3" and "FILE3" all represent different underlying files.

That the approximate sizes ("size" fact) and last modification times ("modify" fact) are the same in all cases might be no more than a coincidence.

It is not suggested that the operators of server-FTPs create an NVFS that stresses the protocols to this extent; however, both user and server implementations must be prepared to deal with such extreme examples.

7.7.9. Example from Another Server

```
C> MlsD
S> 150 File Listing Follows in IMAGE / Binary mode.
D> type=cdir;modify=19990426150227;perm=el; /MISC
D> type=pdir;modify=19791231130000;perm=el; /
D> type=dir;modify=19990426150227;perm=el; CVS
D> type=dir;modify=19990426150228;perm=el; SRC
S> 226 Transfer finished successfully.
C> MlsD src
S> 150 File Listing Follows in IMAGE / Binary mode.
D> type=cdir;modify=19990426150228;perm=el; /MISC/src
D> type=pdir;modify=19990426150227;perm=el; /MISC
D> type=dir;modify=19990426150228;perm=el; CVS
D> type=dir;modify=19990426150228;perm=el; INSTALL
D> type=dir;modify=19990426150230;perm=el; INSTALLI
D> type=dir;modify=19990426150230;perm=el; TREES
S> 226 Transfer finished successfully.
C> MlsD src/install
S> 150 File Listing Follows in IMAGE / Binary mode.
D> type=cdir;modify=19990426150228;perm=el; /MISC/src/install
D> type=pdir;modify=19990426150228;perm=el; /MISC/src
D> type=file;modify=19990406234304;perm=r;size=20059; BOOTPC.C
D> type=file;modify=19980401170153;perm=r;size=278; BOOTPC.H
D> type=file;modify=19990413153736;perm=r;size=54220; BOOTPC.O
D> type=file;modify=19990223044003;perm=r;size=3389; CDROM.C
D> type=file;modify=19990413153739;perm=r;size=30192; CDROM.O
D> type=file;modify=19981119155324;perm=r;size=1055; CHANGELO
D> type=file;modify=19981204171040;perm=r;size=8297; COMMANDS.C
D> type=file;modify=19980508041749;perm=r;size=580; COMMANDS.H
```

```
                                                          ...
D> type=file;modify=19990419052351;perm=r;size=54264; URLMETHO.O
D> type=file;modify=19980218161629;perm=r;size=993; WINDOWS.C
D> type=file;modify=19970912154859;perm=r;size=146; WINDOWS.H
D> type=file;modify=19990413153731;perm=r;size=16812; WINDOWS.O
D> type=file;modify=19990322174959;perm=r;size=129; _CVSIGNO
D> type=file;modify=19990413153640;perm=r;size=82536; _DEPEND
S> 226 Transfer finished successfully.
C> MLst src/install/windows.c
S> 250-Listing src/install/windows.c
S>  type=file;perm=r;size=993; /misc/src/install/windows.c
S> 250 End
S> ftp> mlst SRC/INSTALL/WINDOWS.C
C> MLst SRC/INSTALL/WINDOWS.C
S> 250-Listing SRC/INSTALL/WINDOWS.C
S>  type=file;perm=r;size=993; /misc/SRC/INSTALL/WINDOWS.C
S> 250 End
```

Note that this server gives fully qualified pathnames for the "pdir" and "cdir" entries in MLSD listings. Also notice that this server does, though it is not required to, sort its directory listing outputs. That may be an artifact of the underlying file system access mechanisms of course. Finally notice that the NVFS supported by this server, in contrast to the earlier ones, implements its pathnames in a case independent manner. The server seems to return files using the case in which they were requested, when the name was sent by the client, and otherwise uses an algorithm known only to itself to select the case of the names it returns.

7.7.10. A Server Listing Itself

```
C> MLst f
S> 250-MLST f
S>  Type=dir;Modify=20000710052229;Unique=AAD/AAAABIA; f
S> 250 End
C> CWD f
S> 250 CWD command successful.
C> MLSD
S> 150 Opening ASCII mode data connection for 'MLSD'.
D> Type=cdir;Unique=AAD/AAAABIA; .
D> Type=pdir;Unique=AAD/AAAAAAI; ..
D> Type=file;Size=987;Unique=AAD/AAAABIE; Makefile
D> Type=file;Size=20148;Unique=AAD/AAAABII; conf.c
D> Type=file;Size=11111;Unique=AAD/AAAABIM; extern.h
D> Type=file;Size=38721;Unique=AAD/AAAABIQ; ftpcmd.y
D> Type=file;Size=17922;Unique=AAD/AAAABIU; ftpd.8
D> Type=file;Size=60732;Unique=AAD/AAAABIY; ftpd.c
D> Type=file;Size=3127;Unique=AAD/AAAABIc; logwtmp.c
```

```
D> Type=file;Size=2294;Unique=AAD/AAAABIg; pathnames.h
D> Type=file;Size=7605;Unique=AAD/AAAABIk; popen.c
D> Type=file;Size=9951;Unique=AAD/AAAABIo; ftpd.conf.5
D> Type=file;Size=5023;Unique=AAD/AAAABIs; ftpusers.5
D> Type=file;Size=3547;Unique=AAD/AAAABIw; logutmp.c
D> Type=file;Size=2064;Unique=AAD/AAAABI0; version.h
D> Type=file;Size=20420;Unique=AAD/AAAAAAM; cmds.c
D> Type=file;Size=15864;Unique=AAD/AAAAAAg; ls.c
D> Type=file;Size=2898;Unique=AAD/AAAAAAk; ls.h
D> Type=file;Size=2769;Unique=AAD/AAAAAAo; lsextern.h
D> Type=file;Size=2042;Unique=AAD/AAAAAAs; stat_flags.h
D> Type=file;Size=5708;Unique=AAD/AAAAAAw; cmp.c
D> Type=file;Size=9280;Unique=AAD/AAAAAA0; print.c
D> Type=file;Size=4657;Unique=AAD/AAAAAA4; stat_flags.c
D> Type=file;Size=2664;Unique=AAD/AAAAAA8; util.c
D> Type=file;Size=10383;Unique=AAD/AAAABJ0; ftpd.conf.cat5
D> Type=file;Size=3631;Unique=AAD/AAAABJ4; ftpusers.cat5
D> Type=file;Size=17729;Unique=AAD/AAAABJ8; ftpd.cat8
S> 226 MLSD complete.
```

This examples shows yet another server implementation, showing a listing of its own source code. Note that this implementation does not include the fully qualified path name in its "cdir" and "pdir" entries, nor in the output from "MLST". Also note that the facts requested were modified between the "MLST" and "MLSD" commands, though that exchange has not been shown here.

7.7.11. A Server with a Difference

```
C> PASV
S> 227 Entering Passive Mode (127,0,0,1,255,46)
C> MLSD
S> 150 I tink I tee a trisector tree
D> Type=file;Unique=aaaaafUYqaaa;Perm=rf;Size=15741; x
D> Type=cdir;Unique=aaaaacUYqaaa;Perm=cpmel; /
D> Type=file;Unique=aaaaajUYqaaa;Perm=rf;Size=5760; x4
D> Type=dir;Unique=aaabcaUYqaaa;Perm=elf; sub
D> Type=file;Unique=aaaaagUYqaaa;Perm=rf;Size=8043; x1
D> Type=dir;Unique=aaab8aUYqaaa;Perm=cpmelf; files
D> Type=file;Unique=aaaaahUYqaaa;Perm=rf;Size=4983; x2
D> Type=file;Unique=aaaaaiUYqaaa;Perm=rf;Size=6854; x3
S> 226 That’s all folks...
C> CWD sub
S> 250 CWD command successful.
C> PWD
S> 257 "/sub" is current directory.
C> PASV
S> 227 Entering Passive Mode (127,0,0,1,255,44)
```

```
C> MLSD
S> 150 I tink I tee a trisector tree
D> Type=dir;Unique=aaabceUYqaaa;Perm=elf; dir
D> Type=file;Unique=aaabcbUYqaaa;Perm=rf;Size=0; y1
D> Type=file;Unique=aaabccUYqaaa;Perm=rf;Size=0; y2
D> Type=file;Unique=aaabcdUYqaaa;Perm=rf;Size=0; y3
D> Type=pdir;Unique=aaaaacUYqaaa;Perm=cpmel; /
D> Type=pdir;Unique=aaaaacUYqaaa;Perm=cpmel; ..
D> Type=cdir;Unique=aaabcaUYqaaa;Perm=el; /sub
S> 226 That's all folks...
C> PASV
S> 227 Entering Passive Mode (127,0,0,1,255,42)
C> MLSD dir
S> 150 I tink I tee a trisector tree
D> Type=pdir;Unique=aaabcaUYqaaa;Perm=el; /sub
D> Type=pdir;Unique=aaabcaUYqaaa;Perm=el; ..
D> Type=file;Unique=aaab8cUYqaaa;Perm=r;Size=15039; mlst.c
D> Type=dir;Unique=aaabcfUYqaaa;Perm=el; ect
D> Type=cdir;Unique=aaabceUYqaaa;Perm=el; dir
D> Type=cdir;Unique=aaabceUYqaaa;Perm=el; /sub/dir
D> Type=dir;Unique=aaabchUYqaaa;Perm=el; misc
D> Type=file;Unique=aaab8bUYqaaa;Perm=r;Size=34589; ftpd.c
S> 226 That's all folks...
C> CWD dir/ect
S> 250 CWD command successful.
C> PWD
S> 257 "/sub/dir/ect" is current directory.
C> PASV
S> 227 Entering Passive Mode (127,0,0,1,255,40)
C> MLSD
S> 150 I tink I tee a trisector tree
D> Type=dir;Unique=aaabcgUYqaaa;Perm=el; ory
D> Type=pdir;Unique=aaabceUYqaaa;Perm=el; /sub/dir
D> Type=pdir;Unique=aaabceUYqaaa;Perm=el; ..
D> Type=cdir;Unique=aaabcfUYqaaa;Perm=el; /sub/dir/ect
S> 226 That's all folks...
C> CWD /files
S> 250 CWD command successful.
C> PASV
S> 227 Entering Passive Mode (127,0,0,1,255,36)
C> MLSD
S> 150 I tink I tee a trisector tree
D> Type=cdir;Unique=aaab8aUYqaaa;Perm=cpmel; /files
D> Type=pdir;Unique=aaaaacUYqaaa;Perm=cpmel; /
D> Type=pdir;Unique=aaaaacUYqaaa;Perm=cpmel; ..
D> Type=file;Unique=aaab8cUYqaaa;Perm=rf;Size=15039; mlst.c
D> Type=file;Unique=aaab8bUYqaaa;Perm=rf;Size=34589; ftpd.c
S> 226 That's all folks...
```

```
C> RNFR mlst.c
S> 350 File exists, ready for destination name
C> RNTO list.c
S> 250 RNTO command successful.
C> PASV
S> 227 Entering Passive Mode (127,0,0,1,255,34)
C> MLSD
S> 150 I tink I tee a trisector tree
D> Type=file;Unique=aaab8cUYqaaa;Perm=rf;Size=15039; list.c
D> Type=pdir;Unique=aaaaacUYqaaa;Perm=cpmel; /
D> Type=pdir;Unique=aaaaacUYqaaa;Perm=cpmel; ..
D> Type=file;Unique=aaab8bUYqaaa;Perm=rf;Size=34589; ftpd.c
D> Type=cdir;Unique=aaab8aUYqaaa;Perm=cpmel; /files
S> 226 That's all folks...
```

The server shown here returns its directory listings in seemingly random order, and even seems to modify the order of the directory as its contents change -- perhaps the underlying directory structure is based upon hashing of some kind. Note that the "pdir" and "cdir" entries are interspersed with other entries in the directory. Note also that this server does not show a "pdir" entry when listing the contents of the root directory of the virtual filestore; however, it does however include multiple "cdir" and "pdir" entries when it feels inclined. The server also uses obnoxiously "cute" messages.

7.8. FEAT Response for MLSx

When responding to the FEAT command, a server-FTP process that supports MLST, and MLSD, plus internationalization of pathnames, MUST indicate that this support exists. It does this by including a MLST feature line. As well as indicating the basic support, the MLST feature line indicates which MLST facts are available from the server, and which of those will be returned if no subsequent "OPTS MLST" command is sent.

```
   mlst-feat     = SP "MLST" [SP factlist] CRLF
   factlist      = 1*( factname ["*"] ";" )
```

The initial space shown in the mlst-feat response is that required by the FEAT command, two spaces are not permitted. If no factlist is given, then the server-FTP process is indicating that it supports MLST, but implements no facts. Only pathnames can be returned. This would be a minimal MLST implementation, and useless for most practical purposes. Where the factlist is present, the factnames included indicate the facts supported by the server. Where the optional asterisk appears after a factname, that fact will be included in MLST format responses, until an "OPTS MLST" is given to alter the list of facts returned. After that, subsequent FEAT

commands will return the asterisk to show the facts selected by the most recent "OPTS MLST".

Note that there is no distinct FEAT output for MLSD. The presence of the MLST feature indicates that both MLST and MLSD are supported.

7.8.1. Examples

```
C> Feat
S> 211- Features supported
S>  REST STREAM
S>  MDTM
S>  SIZE
S>  TVFS
S>  UTF8
S>  MLST Type*;Size*;Modify*;Perm*;Unique*;UNIX.mode;UNIX.chgd;X.hidden;
S> 211 End
```

Aside from some features irrelevant here, this server indicates that it supports MLST including several, but not all, standard facts, all of which it will send by default. It also supports two OS dependent facts, and one locally defined fact. The latter three must be requested expressly by the client for this server to supply them.

```
C> Feat
S> 211-Extensions supported:
S>  CLNT
S>  MDTM
S>  MLST type*;size*;modify*;UNIX.mode*;UNIX.owner;UNIX.group;unique;
S>  PASV
S>  REST STREAM
S>  SIZE
S>  TVFS
S>  Compliance Level: 19981201 (IETF mlst-05)
S> 211 End.
```

Again, in addition to some irrelevant features here, this server indicates that it supports MLST, four of the standard facts, one of which ("unique") is not enabled by default, and several OS dependent facts, one of which is provided by the server by default. This server actually supported more OS dependent facts. Others were deleted for the purposes of this document to comply with document formatting restrictions.

```
C> FEAT
S> 211-Features supported
S>  MDTM
S>  MLST Type*;Size*;Modify*;Perm;Unique*;
S>  REST STREAM
S>  SIZE
S>  TVFS
S> 211 End
```

This server has wisely chosen not to implement any OS dependent facts. At the time of writing this document, no such facts have been defined (using the mechanisms of section 10.1) so rational support for them would be difficult at best. All but one of the facts supported by this server are enabled by default.

7.9. OPTS Parameters for MLST

For the MLSx commands, the Client-FTP may specify a list of facts it wishes to be returned in all subsequent MLSx commands until another OPTS MLST command is sent. The format is specified by:

```
   mlst-opts     = "OPTS" SP "MLST"
                   [ SP 1*( factname ";" ) ]
```

By sending the "OPTS MLST" command, the client requests the server to include only the facts listed as arguments to the command in subsequent output from MLSx commands. Facts not included in the "OPTS MLST" command MUST NOT be returned by the server. Facts that are included should be returned for each entry returned from the MLSx command where they meaningfully apply. Facts requested that are not supported, or that are inappropriate to the file or directory being listed should simply be omitted from the MLSx output. This is not an error. Note that where no factname arguments are present, the client is requesting that only the file names be returned. In this case, and in any other case where no facts are included in the result, the space that separates the fact names and their values from the file name is still required. That is, the first character of the output line will be a space, (or two characters will be spaces when the line is returned over the control connection) and the file name will start immediately thereafter.

Clients should note that generating values for some facts can be possible, but very expensive, for some servers. It is generally acceptable to retrieve any of the facts that the server offers as its default set before any "OPTS MLST" command has been given, however clients should use particular caution before requesting any facts not in that set. That is, while other facts may be available from the server, clients should refrain from requesting such facts unless

there is a particular operational requirement for that particular information, which ought be more significant than perhaps simply improving the information displayed to an end user.

Note, there is no "OPTS MLSD" command, the fact names set with the "OPTS MLST" command apply to both MLST and MLSD commands.

Servers are not required to accept "OPTS MLST" commands before authentication of the user-PI, but may choose to permit them.

7.9.1. OPTS MLST Response

The "response-message" from [6] to a successful OPTS MLST command has the following syntax.

```
mlst-opt-resp = "MLST OPTS" [ SP 1*( factname ";" ) ]
```

This defines the "response-message" as used in the "opts-good" message in RFC 2389 [6].

The facts named in the response are those that the server will now include in MLST (and MLSD) response, after the processing of the "OPTS MLST" command. Any facts from the request not supported by the server will be omitted from this response message. If no facts will be included, the list of facts will be empty. Note that the list of facts returned will be the same as those marked by a trailing asterisk ("*") in a subsequent FEAT command response. There is no requirement that the order of the facts returned be the same as that in which they were requested, or that in which they will be listed in a FEAT command response, or that in which facts are returned in MLST responses. The fixed string "MLST OPTS" in the response may be returned in any case, or mixture of cases.

7.9.2. Examples

```
C> Feat
S> 211- Features supported
S>  MLST Type*;Size;Modify*;Perm;Unique;UNIX.mode;UNIX.chgd;X.hidden;
S> 211 End
C> OptS Mlst Type;UNIX.mode;Perm;
S> 200 MLST OPTS Type;Perm;UNIX.mode;
C> Feat
S> 211- Features supported
S>  MLST Type*;Size;Modify;Perm*;Unique;UNIX.mode*;UNIX.chgd;X.hidden;
S> 211 End
C> opts MLst lang;type;charset;create;
S> 200 MLST OPTS Type;
C> Feat
```

```
S> 211- Features supported
S>  MLST Type*;Size;Modify;Perm;Unique;UNIX.mode;UNIX.chgd;X.hidden;
S> 211 End
C> OPTS mlst size;frogs;
S> 200 MLST OPTS Size;
C> Feat
S> 211- Features supported
S>  MLST Type;Size*;Modify;Perm;Unique;UNIX.mode;UNIX.chgd;X.hidden;
S> 211 End
C> opts MLst unique type;
S> 501 Invalid MLST options
C> Feat
S> 211- Features supported
S>  MLST Type;Size*;Modify;Perm;Unique;UNIX.mode;UNIX.chgd;X.hidden;
S> 211 End
```

For the purposes of this example, features other than MLST have been deleted from the output to avoid clutter. The example shows the initial default feature output for MLST. The facts requested are then changed by the client. The first change shows facts that are available from the server being selected. Subsequent FEAT output shows the altered features as being returned. The client then attempts to select some standard features that the server does not support. This is not an error, however the server simply ignores the requests for unsupported features, as the FEAT output that follows shows. Then, the client attempts to request a non-standard, and unsupported, feature. The server ignores that, and selects only the supported features requested. Lastly, the client sends a request containing a syntax error (spaces cannot appear in the factlist.) The server-FTP sends an error response and completely ignores the request, leaving the fact set selected as it had been previously.

Note that in all cases, except the error response, the response lists the facts that have been selected.

```
C> Feat
S> 211- Features supported
S>  MLST Type*;Size*;Modify*;Perm*;Unique*;UNIX.mode;UNIX.chgd;X.hidden;
S> 211 End
C> Opts MLST
S> 200 MLST OPTS
C> Feat
S> 211- Features supported
S>  MLST Type;Size;Modify;Perm;Unique;UNIX.mode;UNIX.chgd;X.hidden;
S> 211 End
C> MLst tmp
S> 250- Listing tmp
S>   /tmp
```

```
S> 250 End
C> OPTS mlst unique;size;
S> 200 MLST OPTS Size;Unique;
C>  MLst tmp
S> 250- Listing tmp
S>  Unique=keVO1+YZ5; /tmp
S> 250 End
C> OPTS mlst unique;type;modify;
S> 200 MLST OPTS Type;Modify;Unique;
C> MLst tmp
S> 250- Listing tmp
S>  Type=dir;Modify=19990930152225;Unique=keVO1+YZ5; /tmp
S> 250 End
C> OPTS mlst fish;cakes;
S> 200 MLST OPTS
C> MLst tmp
S> 250- Listing tmp
S>   /tmp
S> 250 End
C> OptS Mlst Modify;Unique;
S> 200 MLST OPTS Modify;Unique;
C> MLst tmp
S> 250- Listing tmp
S>  Modify=19990930152225;Unique=keVO1+YZ5; /tmp
S> 250 End
C> opts MLst fish cakes;
S> 501 Invalid MLST options
C> MLst tmp
S> 250- Listing tmp
S>  Modify=19990930152225;Unique=keVO1+YZ5; /tmp
S> 250 End
```

This example shows the effect of changing the facts requested upon subsequent MLST commands. Notice that a syntax error leaves the set of selected facts unchanged. Also notice exactly two spaces preceding the pathname when no facts were selected, either deliberately, or because none of the facts requested were available.

8. Impact on Other FTP Commands

Along with the introduction of MLST, traditional FTP commands must be extended to allow for the use of more than US-ASCII [1] or EBCDIC character sets. In general, the support of MLST requires support for arbitrary character sets wherever file names and directory names are allowed. This applies equally to both arguments given to the following commands and to the replies from them, as appropriate.

APPE	RMD
CWD	RNFR
DELE	RNTO
MKD	STAT
PWD	STOR
RETR	STOU

The arguments to all of these commands should be processed the same way that MLST commands and responses are processed with respect to handling embedded spaces, CRs and NULs. See section 2.2.

9. Character Sets and Internationalization

FTP commands are protocol elements, and are always expressed in ASCII. FTP responses are composed of the numeric code, which is a protocol element, and a message, which is often expected to convey information to the user. It is not expected that users normally interact directly with the protocol elements, rather the user-FTP process constructs the commands, and interprets the results, in the manner best suited for the particular user. Explanatory text in responses generally has no particular meaning to the protocol. The numeric codes provide all necessary information. Server-PIs are free to provide the text in any language that can be adequately represented in ASCII, or where an alternative language and representation has been negotiated (see [7]) in that language and representation.

Pathnames are expected to be encoded in UTF-8 allowing essentially any character to be represented in a pathname. Meaningful pathnames are defined by the server NVFS.

No restrictions at all are placed upon the contents of files transferred using the FTP protocols. Unless the "media-type" fact is provided in a MLSx response nor is any advice given here that would allow determining the content type. That information is assumed to be obtained via other means.

10. IANA Considerations

This specification makes use of some lists of values currently maintained by the IANA, and creates two new lists for the IANA to maintain. It does not add any values to any existing registries.

The existing IANA registries used by this specification are modified using mechanisms specified elsewhere.

10.1. The OS-Specific Fact Registry

A registry of OS specific fact names shall be maintained by the IANA. The OS names for the OS portion of the fact name must be taken from the IANA's list of registered OS names. To add a fact name to this OS specific registry of OS specific facts, an applicant must send to the IANA a request, in which is specified the OS name, the OS specific fact name, a definition of the syntax of the fact value, which must conform to the syntax of a token as given in this document, and a specification of the semantics to be associated with the particular fact and its values. Upon receipt of such an application, and if the combination of OS name and OS specific fact name has not been previously defined, the IANA will add the specification to the registry.

Any examples of OS specific facts found in this document are to be treated as examples of possible OS specific facts, and do not form a part of the IANA's registry merely because of being included in this document.

10.2. The OS-Specific Filetype Registry

A registry of OS specific file types shall be maintained by the IANA. The OS names for the OS portion of the fact name must be taken from the IANA's list of registered OS names. To add a file type to this OS specific registry of OS specific file types, an applicant must send to the IANA a request, in which is specified the OS name, the OS specific file type, a definition of the syntax of the fact value, which must conform to the syntax of a token as given in this document, and a specification of the semantics to be associated with the particular fact and its values. Upon receipt of such an application, and if the combination of OS name and OS specific file type has not been previously defined, the IANA will add the specification to the registry.

Any examples of OS specific file types found in this document are to be treated as potential OS specific file types only, and do not form a part of the IANA's registry merely because of being included in this document.

11. Security Considerations

This memo does not directly concern security. It is not believed that any of the mechanisms documented here impact in any particular way upon the security of FTP.

Implementing the SIZE command, and perhaps some of the facts of the MLSx commands, may impose a considerable load on the server, which could lead to denial of service attacks. Servers have, however, implemented this for many years, without significant reported difficulties.

The server-FTP should take care not to reveal sensitive information about files to unauthorised parties. In particular, some underlying filesystems provide a file identifier that, if known, can allow many of the filesystem protection mechanisms to be by-passed. That identifier would not be a suitable choice to use as the basis of the value of the unique fact.

The FEAT and OPTS commands may be issued before the FTP authentication has occurred [6]. This allows unauthenticated clients to determine which of the features defined here are supported, and to negotiate the fact list for MLSx output. No actual MLSx commands may be issued however, and no problems with permitting the selection of the format prior to authentication are foreseen.

A general discussion of issues related to the security of FTP can be found in [13].

12. Normative References

[1] Coded Character Set--7-bit American Standard Code for Information Interchange, ANSI X3.4-1986.

[2] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 3629, November 2003.

[3] Postel, J. and J. Reynolds, "File Transfer Protocol (FTP)", STD 9, RFC 959, October 1985.

[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[5] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[6] Hethmon, P. and R. Elz, "Feature negotiation mechanism for the File Transfer Protocol", RFC 2389, August 1998.

[7] Curtin, B., "Internationalization of the File Transfer Protocol", RFC 2640, July 1999.

[8] Postel, J. and J. Reynolds, "Telnet protocol Specification", STD 8, RFC 854, May 1983.

[9] Braden, R,. "Requirements for Internet Hosts -- Application and Support", STD 3, RFC 1123, October 1989.

[10] ISO/IEC 10646-1:1993 "Universal multiple-octet coded character set (UCS) -- Part 1: Architecture and basic multilingual plane", International Standard -- Information Technology, 1993.

[11] Internet Assigned Numbers Authority. http://www.iana.org Email: iana@iana.org.

[12] Phillips, A. and M. Davis, "Tags for Identifying Languages", BCP 47, RFC 4646, September 2006.

[13] Allman, M. and S. Ostermann, "FTP Security Considerations" RFC 2577, May 1999.

Acknowledgments

This document is a product of the FTPEXT working group of the IETF.

The following people are among those who have contributed to this document:

Alex Belits
D. J. Bernstein
Dave Cridland
Martin J. Duerst
Bill Fenner (and the rest of the IESG)
Paul Ford-Hutchinson
Mike Gleason
Mark Harris
Stephen Head
Alun Jones
Andrew Main
James Matthews
Luke Mewburn
Jan Mikkelsen
Keith Moore
Buz Owen
Mark Symons
Stephen Tihor
and the entire FTPEXT working group of the IETF.

Apologies are offered to any inadvertently omitted.

The description of the modifications to the REST command and the MDTM and SIZE commands comes from a set of modifications suggested for STD 9, RFC 959 by Rick Adams in 1989. A document containing just those commands, edited by David Borman, has been merged with this document.

Mike Gleason, Alun Jones and Luke Mewburn provided access to FTP servers used in some of the examples.

All of the examples in this document are taken from actual client/server exchanges, though some have been edited for brevity, or to meet document formatting requirements.

RFC Editor Note:

Several of the examples in this document exceed the RFC standard line length of 72 characters. Since this document is a standards-track result of an IETF working group and is important to an IETF sub-community, the RFC Editor is publishing it with the margin violations. This is not a precedent.

Author's Address

Paul Hethmon
Hethmon Software
10420 Jackson Oaks Way, Suite 201
Knoxville, TN 37922

EMail: phethmon@hethmon.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.