*:96 Internet application layer protocols and standards

Compendium 2: Allowed during the exam

Last revision: 1 Apr 2003

FTP
RFC 959: File Transfer Protocol (FTP) 253-287

Cookies
RFC 2109: HTTP State Management Mechanism 288-298

Usenet News Message Format
RFC 1036: Standard for Interchange of USENET Messages 299-308

HTTP
RFC 2068: Hypertext Transfer Protocol HTTP 1.1 309-389

NNTP
RFC 977: Network News Transfer Protocol (NNTP) 390-403

URL
RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax 425-444

Port Numbers
IANA Register of Port Numbers 445-460

Media Types
IANA Register of Media Types 461-468

The documents are not ordered in a suitable order for reading them,
see compendium 0 page14-17

Network Working Group J. Postel
Request for Comments: 959 J. Reynolds
ISI
Obsoletes RFC: 765 (IEN 149) October 1985

FILE TRANSFER PROTOCOL (FTP)

Status of this Memo

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

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

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

Note that this specification is compatible with the previous edition.

1. INTRODUCTION

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

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

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

2. OVERVIEW

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

2.1. HISTORY

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

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

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

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

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

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

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

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

CDUP - Change to Parent Directory

SMNT - Structure Mount

STOU - Store Unique

RMD - Remove Directory

MKD - Make Directory

PWD - Print Directory

SYST - System

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

2.2. TERMINOLOGY

ASCII

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

access controls

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

byte size

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

control connection

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

data connection

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

data port

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

DTP

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

End-of-Line

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

EOF

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

EOR

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

error recovery

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

FTP commands

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

file

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

mode

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

NVT

The Network Virtual Terminal as defined in the Telnet Protocol.

NVFS

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

page

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

pathname

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

PI

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

record

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

reply

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

server-DTP

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

server-FTP process

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

server-PI

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

type

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

user

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

user-DTP

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

user-FTP process

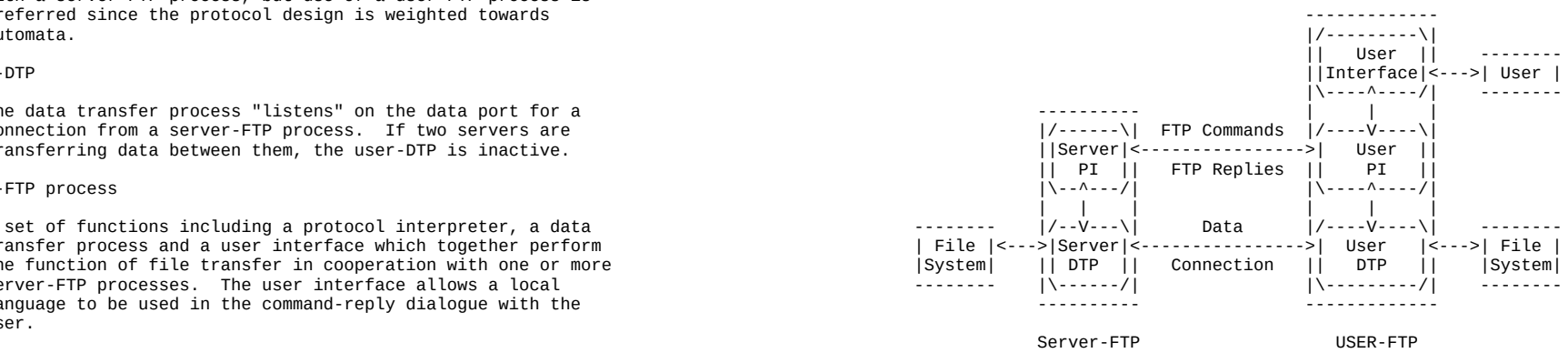
A set of functions including a protocol interpreter, a data transfer process and a user interface which together perform the function of file transfer in cooperation with one or more server-FTP processes. The user interface allows a local language to be used in the command-reply dialogue with the user.

user-PI

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

2.3. THE FTP MODEL

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



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

Figure 1 Model for FTP Use

In the model described in Figure 1, the user-protocol interpreter initiates the control connection. The control connection follows the Telnet protocol. At the initiation of the user, standard FTP commands are generated by the user-PI and transmitted to the server process via the control connection. (The user may establish a direct control connection to the server-FTP, from a TAC terminal for example, and generate standard FTP commands independently, bypassing the user-FTP process.) Standard replies are sent from the server-PI to the user-PI over the control connection in response to the commands.

The FTP commands specify the parameters for the data connection (data port, transfer mode, representation type, and structure) and the nature of file system operation (store, retrieve, append, delete, etc.). The user-DTP or its designate should "listen" on the specified data port, and the server initiate the data connection and data transfer in accordance with the specified parameters. It should be noted that the data port need not be in

the same host that initiates the FTP commands via the control connection, but the user or the user-FTP process must ensure a "listen" on the specified data port. It ought to also be noted that the data connection may be used for simultaneous sending and receiving.

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

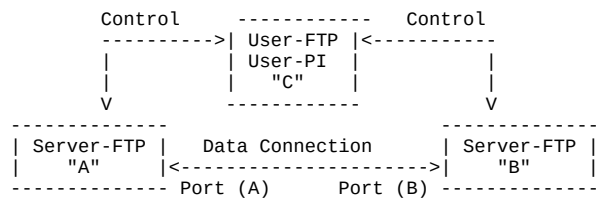


Figure 2

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

The Relationship between FTP and Telnet:

The FTP uses the Telnet protocol on the control connection. This can be achieved in two ways: first, the user-PI or the server-PI may implement the rules of the Telnet Protocol directly in their own procedures; or, second, the user-PI or the server-PI may make use of the existing Telnet module in the system.

Ease of implementaion, sharing code, and modular programming argue for the second approach. Efficiency and independence

argue for the first approach. In practice, FTP relies on very little of the Telnet Protocol, so the first approach does not necessarily involve a large amount of code.

3. DATA TRANSFER FUNCTIONS

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

3.1. DATA REPRESENTATION AND STORAGE

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

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

that FTP provides for very limited data type representations. Transformations desired beyond this limited capability should be performed by the user directly.

3.1.1. DATA TYPES

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

3.1.1.1. ASCII TYPE

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

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

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

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

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

3.1.1.2. EBCDIC TYPE

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

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

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

3.1.1.3. IMAGE TYPE

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

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

3.1.1.4. LOCAL TYPE

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

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

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

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

3.1.1.5. FORMAT CONTROL

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

A character file may be transferred to a host for one of three purposes: for printing, for storage and later retrieval, or for processing. If a file is sent for printing, the receiving host must know how the vertical format control is represented. In the second case, it must be possible to store a file at a host and then retrieve it later in exactly the same form. Finally, it should be possible to move a file from one host to another and process the file at the second host without undue trouble. A single ASCII or EBCDIC format does not satisfy all these conditions. Therefore, these types have a second parameter specifying one of the following three formats:

3.1.1.5.1. NON PRINT

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

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

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

3.1.1.5.2. TELNET FORMAT CONTROLS

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

3.1.1.5.2. CARRIAGE CONTROL (ASA)

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

The ASA Standard specifies the following control characters:

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

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

3.1.2. DATA STRUCTURES

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

file-structure,	where there is no internal structure and the file is considered to be a continuous sequence of data bytes,
record-structure,	where the file is made up of sequential records,
and page-structure,	where the file is made up of independent indexed pages.

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

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

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

In the case of a file being sent with file-structure to a record-oriented host, there exists the question of what criteria the host should use to divide the file into records which can be processed locally. If this division is necessary, the FTP implementation should use the end-of-line sequence,

<CRLF> for ASCII, or <NL> for EBCDIC text files, as the delimiter. If an FTP implementation adopts this technique, it must be prepared to reverse the transformation if the file is retrieved with file-structure.

3.1.2.1. FILE STRUCTURE

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

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

3.1.2.2. RECORD STRUCTURE

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

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

3.1.2.3. PAGE STRUCTURE

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

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

Header Length

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

Page Index

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

Data Length

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

Page Type

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

0 = Last Page

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

1 = Simple Page

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

2 = Descriptor Page

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

3 = Access Controlled Page

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

Optional Fields

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

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

A note of caution about parameters: a file must be stored and retrieved with the same parameters if the retrieved version is to

be identical to the version originally transmitted. Conversely, FTP implementations must return a file identical to the original if the parameters used to store and retrieve a file are the same.

3.2. ESTABLISHING DATA CONNECTIONS

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

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

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

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

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

In general, it is the server's responsibility to maintain the data connection--to initiate it and to close it. The exception to this

is when the user-DTP is sending the data in a transfer mode that requires the connection to be closed to indicate EOF. The server MUST close the data connection under the following conditions:

1. The server has completed sending data in a transfer mode that requires a close to indicate EOF.

2. The server receives an ABORT command from the user.

3. The port specification is changed by a command from the user.

4. The control connection is closed legally or otherwise.

5. An irrecoverable error condition occurs.

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

3.3. DATA CONNECTION MANAGEMENT

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

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

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

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

A comment on transfer modes. The stream transfer mode is

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

3.4. TRANSMISSION MODES

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

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

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

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

The following transmission modes are defined in FTP:

3.4.1. STREAM MODE

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

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

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

3.4.2. BLOCK MODE

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

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

Block Header

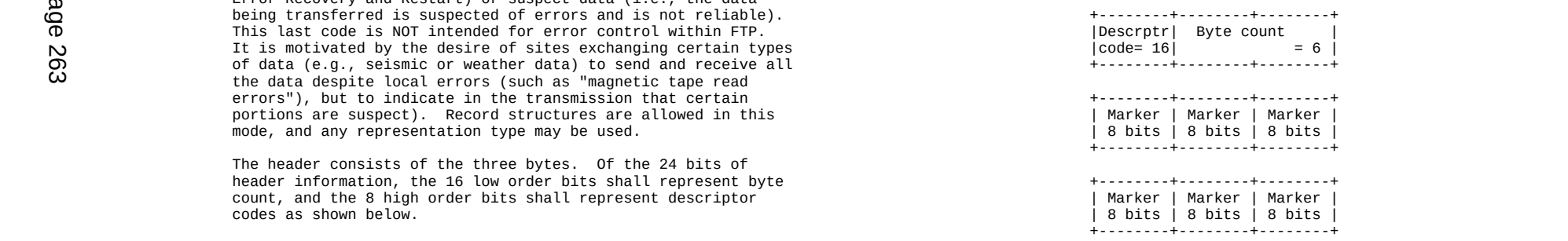
The descriptor codes are indicated by bit flags in the descriptor byte. Four codes have been assigned, where each code number is the decimal value of the corresponding bit in the byte.

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

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

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

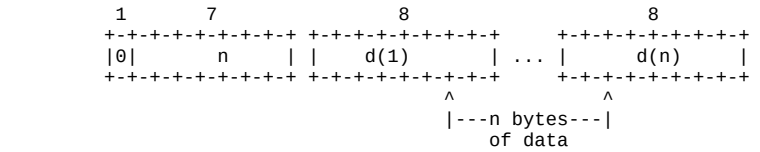
For example, to transmit a six-character marker, the following would be sent:



3.4.3. COMPRESSED MODE

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

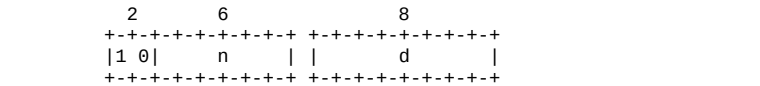
Byte string:



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

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

Replicated Byte:



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

Filler String:

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

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

3.5. ERROR RECOVERY AND RESTART

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

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

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

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

connection and is immediately followed by the command (such as RETR, STOR or LIST) which was being executed when the system failure occurred.

4. FILE TRANSFER FUNCTIONS

The communication channel from the user-PI to the server-PI is established as a TCP connection from the user to the standard server port. The user protocol interpreter is responsible for sending FTP commands and interpreting the replies received; the server-PI interprets commands, sends replies and directs its DTP to set up the data connection and transfer the data. If the second party to the data transfer (the passive transfer process) is the user-DTP, then it is governed through the internal protocol of the user-FTP host; if it is a second server-DTP, then it is governed by its PI on command from the user-PI. The FTP replies are discussed in the next section. In the description of a few of the commands in this section, it is helpful to be explicit about the possible replies.

4.1. FTP COMMANDS

4.1.1. ACCESS CONTROL COMMANDS

The following commands specify access control identifiers (command codes are shown in parentheses).

USER NAME (USER)

The argument field is a Telnet string identifying the user. The user identification is that which is required by the server for access to its file system. This command will normally be the first command transmitted by the user after the control connections are made (some servers may require this). Additional identification information in the form of a password and/or an account command may also be required by some servers. Servers may allow a new USER command to be entered at any point in order to change the access control and/or accounting information. This has the effect of flushing any user, password, and account information already supplied and beginning the login sequence again. All transfer parameters are unchanged and any file transfer in progress is completed under the old access control parameters.

PASSWORD (PASS)

The argument field is a Telnet string specifying the user's password. This command must be immediately preceded by the user name command, and, for some sites, completes the user's identification for access control. Since password information is quite sensitive, it is desirable in general to "mask" it or suppress typeout. It appears that the server has no foolproof way to achieve this. It is therefore the responsibility of the user-FTP process to hide the sensitive password information.

ACCOUNT (ACCT)

The argument field is a Telnet string identifying the user's account. The command is not necessarily related to the USER command, as some sites may require an account for login and others only for specific access, such as storing files. In the latter case the command may arrive at any time.

There are reply codes to differentiate these cases for the automation: when account information is required for login, the response to a successful PASSword command is reply code 332. On the other hand, if account information is NOT required for login, the reply to a successful PASSword command is 230; and if the account information is needed for a command issued later in the dialogue, the server should return a 332 or 532 reply depending on whether it stores (pending receipt of the ACCounT command) or discards the command, respectively.

CHANGE WORKING DIRECTORY (CWD)

This command allows the user to work with a different directory or dataset for file storage or retrieval without altering his login or accounting information. Transfer parameters are similarly unchanged. The argument is a pathname specifying a directory or other system dependent file group designator.

CHANGE TO PARENT DIRECTORY (CDUP)

This command is a special case of CWD, and is included to simplify the implementation of programs for transferring directory trees between operating systems having different

syntaxes for naming the parent directory. The reply codes shall be identical to the reply codes of CWD. See Appendix II for further details.

STRUCTURE MOUNT (SMNT)

This command allows the user to mount a different file system data structure without altering his login or accounting information. Transfer parameters are similarly unchanged. The argument is a pathname specifying a directory or other system dependent file group designator.

REINITIALIZE (REIN)

This command terminates a USER, flushing all I/O and account information, except to allow any transfer in progress to be completed. All parameters are reset to the default settings and the control connection is left open. This is identical to the state in which a user finds himself immediately after the control connection is opened. A USER command may be expected to follow.

LOGOUT (QUIT)

This command terminates a USER and if file transfer is not in progress, the server closes the control connection. If file transfer is in progress, the connection will remain open for result response and the server will then close it. If the user-process is transferring files for several USERs but does not wish to close and then reopen connections for each, then the REIN command should be used instead of QUIT.

An unexpected close on the control connection will cause the server to take the effective action of an abort (ABOR) and a logout (QUIT).

4.1.2. TRANSFER PARAMETER COMMANDS

All data transfer parameters have default values, and the commands specifying data transfer parameters are required only if the default parameter values are to be changed. The default value is the last specified value, or if no value has been specified, the standard default value is as stated here. This implies that the server must "remember" the applicable default values. The commands may be in any order except that they must precede the FTP service request. The following commands specify data transfer parameters:

DATA PORT (PORT)

The argument is a HOST-PORT specification for the data port to be used in data connection. There are defaults for both the user and server data ports, and under normal circumstances this command and its reply are not needed. If this command is used, the argument is the concatenation of a 32-bit internet host address and a 16-bit TCP port address. This address information is broken into 8-bit fields and the value of each field is transmitted as a decimal number (in character string representation). The fields are separated by commas. A port command would be:

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

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

PASSIVE (PASV)

This command requests the server-DTP to "listen" on a data port (which is not its default data port) and to wait for a connection rather than initiate one upon receipt of a transfer command. The response to this command includes the host and port address this server is listening on.

REPRESENTATION TYPE (TYPE)

The argument specifies the representation type as described in the Section on Data Representation and Storage. Several types take a second parameter. The first parameter is denoted by a single Telnet character, as is the second Format parameter for ASCII and EBCDIC; the second parameter for local byte is a decimal integer to indicate Bytesize. The parameters are separated by a <SP> (Space, ASCII code 32).

The following codes are assigned for type:

I - Image

L <byte size> - Local byte Byte size

The default representation type is ASCII Non-print. If the Format parameter is changed, and later just the first argument is changed, Format then returns to the Non-print default.

FILE STRUCTURE (STRU)

The argument is a single Telnet character code specifying file structure described in the Section on Data Representation and Storage.

The following codes are assigned for structure:

F - File (no record structure)
R - Record structure
P - Page structure

The default structure is File.

TRANSFER MODE (MODE)

The argument is a single Telnet character code specifying the data transfer modes described in the Section on Transmission Modes.

The following codes are assigned for transfer modes:

S - Stream
B - Block
C - Compressed

The default transfer mode is Stream.

4.1.3. FTP SERVICE COMMANDS

The FTP service commands define the file transfer or the file system function requested by the user. The argument of an FTP service command will normally be a pathname. The syntax of pathnames must conform to server site conventions (with standard defaults applicable), and the language conventions of the control connection. The suggested default handling is to use the last specified device, directory or file name, or the standard default defined for local users. The commands may be in any order except that a "rename from" command must be followed by a "rename to" command and the restart command must be followed by the interrupted service command (e.g., STOR or RETR). The data, when transferred in response to FTP service

commands, shall always be sent over the data connection, except for certain informative replies. The following commands specify FTP service requests:

RETRIEVE (RETR)

This command causes the server-DTP to transfer a copy of the file, specified in the pathname, to the server- or user-DTP at the other end of the data connection. The status and contents of the file at the server site shall be unaffected.

STORE (STOR)

This command causes the server-DTP to accept the data transferred via the data connection and to store the data as a file at the server site. If the file specified in the pathname exists at the server site, then its contents shall be replaced by the data being transferred. A new file is created at the server site if the file specified in the pathname does not already exist.

STORE UNIQUE (STOU)

This command behaves like STOR except that the resultant file is to be created in the current directory under a name unique to that directory. The 250 Transfer Started response must include the name generated.

APPEND (with create) (APPE)

This command causes the server-DTP to accept the data transferred via the data connection and to store the data in a file at the server site. If the file specified in the pathname exists at the server site, then the data shall be appended to that file; otherwise the file specified in the pathname shall be created at the server site.

ALLOCATE (ALLO)

This command may be required by some servers to reserve sufficient storage to accommodate the new file to be transferred. The argument shall be a decimal integer representing the number of bytes (using the logical byte size) of storage to be reserved for the file. For files sent with record or page structure a maximum record or page size (in logical bytes) might also be necessary; this is indicated by a decimal integer in a second argument field of

the command. This second argument is optional, but when present should be separated from the first by the three Telnet characters <SP> R <SP>. This command shall be followed by a STORe or APPEnd command. The ALLO command should be treated as a NOOP (no operation) by those servers which do not require that the maximum size of the file be declared beforehand, and those servers interested in only the maximum record or page size should accept a dummy value in the first argument and ignore it.

RESTART (REST)

The argument field represents the server marker at which file transfer is to be restarted. This command does not cause file transfer but skips over the file to the specified data checkpoint. This command shall be immediately followed by the appropriate FTP service command which shall cause file transfer to resume.

RENAME FROM (RNFR)

This command specifies the old pathname of the file which is to be renamed. This command must be immediately followed by a "rename to" command specifying the new file pathname.

RENAME TO (RNTO)

This command specifies the new pathname of the file specified in the immediately preceding "rename from" command. Together the two commands cause a file to be renamed.

ABORT (ABOR)

This command tells the server to abort the previous FTP service command and any associated transfer of data. The abort command may require "special action", as discussed in the Section on FTP Commands, to force recognition by the server. No action is to be taken if the previous command has been completed (including data transfer). The control connection is not to be closed by the server, but the data connection must be closed.

There are two cases for the server upon receipt of this command: (1) the FTP service command was already completed, or (2) the FTP service command is still in progress.

In the first case, the server closes the data connection (if it is open) and responds with a 226 reply, indicating that the abort command was successfully processed.

In the second case, the server aborts the FTP service in progress and closes the data connection, returning a 426 reply to indicate that the service request terminated abnormally. The server then sends a 226 reply, indicating that the abort command was successfully processed.

DELETE (DELE)

This command causes the file specified in the pathname to be deleted at the server site. If an extra level of protection is desired (such as the query, "Do you really wish to delete?"), it should be provided by the user-FTP process.

REMOVE DIRECTORY (RMD)

This command causes the directory specified in the pathname to be removed as a directory (if the pathname is absolute) or as a subdirectory of the current working directory (if the pathname is relative). See Appendix II.

MAKE DIRECTORY (MKD)

This command causes the directory specified in the pathname to be created as a directory (if the pathname is absolute) or as a subdirectory of the current working directory (if the pathname is relative). See Appendix II.

PRINT WORKING DIRECTORY (PWD)

This command causes the name of the current working directory to be returned in the reply. See Appendix II.

LIST (LIST)

This command causes a list to be sent from the server to the passive DTP. If the pathname specifies a directory or other group of files, the server should transfer a list of files in the specified directory. If the pathname specifies a file then the server should send current information on the file. A null argument implies the user's current working or default directory. The data transfer is over the data connection in type ASCII or type EBCDIC. (The user must

ensure that the TYPE is appropriately ASCII or EBCDIC). Since the information on a file may vary widely from system to system, this information may be hard to use automatically in a program, but may be quite useful to a human user.

NAME LIST (NLST)

This command causes a directory listing to be sent from server to user site. The pathname should specify a directory or other system-specific file group descriptor; a null argument implies the current directory. The server will return a stream of names of files and no other information. The data will be transferred in ASCII or EBCDIC type over the data connection as valid pathname strings separated by <CRLF> or <NL>. (Again the user must ensure that the TYPE is correct.) This command is intended to return information that can be used by a program to further process the files automatically. For example, in the implementation of a "multiple get" function.

SITE PARAMETERS (SITE)

This command is used by the server to provide services specific to his system that are essential to file transfer but not sufficiently universal to be included as commands in the protocol. The nature of these services and the specification of their syntax can be stated in a reply to the HELP SITE command.

SYSTEM (SYST)

This command is used to find out the type of operating system at the server. The reply shall have as its first word one of the system names listed in the current version of the Assigned Numbers document [4].

STATUS (STAT)

This command shall cause a status response to be sent over the control connection in the form of a reply. The command may be sent during a file transfer (along with the Telnet IP and Synch signals--see the Section on FTP Commands) in which case the server will respond with the status of the operation in progress, or it may be sent between file transfers. In the latter case, the command may have an argument field. If the argument is a pathname, the command is analogous to the "list" command except that data shall be

transferred over the control connection. If a partial pathname is given, the server may respond with a list of file names or attributes associated with that specification. If no argument is given, the server should return general status information about the server FTP process. This should include current values of all transfer parameters and the status of connections.

HELP (HELP)

This command shall cause the server to send helpful information regarding its implementation status over the control connection to the user. The command may take an argument (e.g., any command name) and return more specific information as a response. The reply is type 211 or 214. It is suggested that HELP be allowed before entering a USER command. The server may use this reply to specify site-dependent parameters, e.g., in response to HELP SITE.

NOOP (NOOP)

This command does not affect any parameters or previously entered commands. It specifies no action other than that the server send an OK reply.

The File Transfer Protocol follows the specifications of the Telnet protocol for all communications over the control connection. Since the language used for Telnet communication may be a negotiated option, all references in the next two sections will be to the "Telnet language" and the corresponding "Telnet end-of-line code". Currently, one may take these to mean NVT-ASCII and <CRLF>. No other specifications of the Telnet protocol will be cited.

FTP commands are "Telnet strings" terminated by the "Telnet end of line code". The command codes themselves are alphabetic characters terminated by the character <SP> (Space) if parameters follow and Telnet-EOL otherwise. The command codes and the semantics of commands are described in this section; the detailed syntax of commands is specified in the Section on Commands, the reply sequences are discussed in the Section on Sequencing of Commands and Replies, and scenarios illustrating the use of commands are provided in the Section on Typical FTP Scenarios.

FTP commands may be partitioned as those specifying access-control identifiers, data transfer parameters, or FTP service requests. Certain commands (such as ABOR, STAT, QUIT) may be sent over the control connection while a data transfer is in progress. Some

servers may not be able to monitor the control and data connections simultaneously, in which case some special action will be necessary to get the server's attention. The following ordered format is tentatively recommended:

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

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

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

4. Server PI, after receiving "IP", scans the Telnet stream for EXACTLY ONE FTP command.

(For other servers this may not be necessary but the actions listed above should have no unusual effect.)

4.2. FTP REPLIES

Replies to File Transfer Protocol commands are devised to ensure the synchronization of requests and actions in the process of file transfer, and to guarantee that the user process always knows the state of the Server. Every command must generate at least one reply, although there may be more than one; in the latter case, the multiple replies must be easily distinguished. In addition, some commands occur in sequential groups, such as USER, PASS and ACCT, or RNFR and RNTO. The replies show the existence of an intermediate state if all preceding commands have been successful. A failure at any point in the sequence necessitates the repetition of the entire sequence from the beginning.

The details of the command-reply sequence are made explicit in a set of state diagrams below.

An FTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is intended for the human user. It is intended that the three digits contain enough encoded information that the user-process (the User-PI) will not need to examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be server-dependent, so there are likely to be varying texts for each reply code.

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

<SP>, followed by one line of text (where some maximum line length has been specified), and terminated by the Telnet end-of-line code. There will be cases however, where the text is longer than a single line. In these cases the complete text must be bracketed so the User-process knows when it may stop reading the reply (i.e. stop processing input on the control connection) and go do other things. This requires a special format on the first line to indicate that more than one line is coming, and another on the last line to designate it as the last. At least one of these must contain the appropriate reply code to indicate the state of the transaction. To satisfy all factions, it was decided that both the first and last line codes should be the same.

Thus the format for multi-line replies is that the first line will begin with the exact required reply code, followed immediately by a Hyphen, "-" (also known as Minus), followed by text. The last line will begin with the same code, followed immediately by Space <SP>, optionally some text, and the Telnet end-of-line code.

For example:

```
123-First line
Second line
  234 A line beginning with numbers
123 The last line
```

The user-process then simply needs to search for the second occurrence of the same reply code, followed by <SP> (Space), at the beginning of a line, and ignore all intermediary lines. If an intermediary line begins with a 3-digit number, the Server must pad the front to avoid confusion.

This scheme allows standard system routines to be used for reply information (such as for the STAT reply), with "artificial" first and last lines tacked on. In rare cases where these routines are able to generate three digits and a Space at the beginning of any line, the beginning of each text line should be offset by some neutral text, like Space.

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

The three digits of the reply each have a special significance. This is intended to allow a range of very simple to very sophisticated responses by the user-process. The first digit denotes whether the response is good, bad or incomplete. (Referring to the state diagram), an unsophisticated user-process will be able to determine its next action (proceed as planned,

redo, retrench, etc.) by simply examining this first digit. A user-process that wants to know approximately what kind of error occurred (e.g. file system error, command syntax error) may examine the second digit, reserving the third digit for the finest gradation of information (e.g., RNTO command without a preceding RNFR).

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

1yz Positive Preliminary reply

The requested action is being initiated; expect another reply before proceeding with a new command. (The user-process sending another command before the completion reply would be in violation of protocol; but server-FTP processes should queue any commands that arrive while a preceding command is in progress.) This type of reply can be used to indicate that the command was accepted and the user-process may now pay attention to the data connections, for implementations where simultaneous monitoring is difficult. The server-FTP process may send at most, one 1yz reply per command.

2yz Positive Completion reply

The requested action has been successfully completed. A new request may be initiated.

3yz Positive Intermediate reply

The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The user should send another command specifying this information. This reply is used in command sequence groups.

4yz Transient Negative Completion reply

The command was not accepted and the requested action did not take place, but the error condition is temporary and the action may be requested again. The user should return to the beginning of the command sequence, if any. It is difficult to assign a meaning to "transient", particularly when two distinct sites (Server- and User-processes) have to agree on the interpretation. Each reply in the 4yz category might have a slightly different time value, but the intent is that the

user-process is encouraged to try again. A rule of thumb in determining if a reply fits into the 4yz or the 5yz (Permanent Negative) category is that replies are 4yz if the commands can be repeated without any change in command form or in properties of the User or Server (e.g., the command is spelled the same with the same arguments used; the user does not change his file access or user name; the server does not put up a new implementation.)

5yz Permanent Negative Completion reply

The command was not accepted and the requested action did not take place. The User-process is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct his User-process to reinitiate the command sequence by direct action at some point in the future (e.g., after the spelling has been changed, or the user has altered his directory status.)

The following function groupings are encoded in the second digit:

x0z Syntax - These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, unimplemented or superfluous commands.

x1z Information - These are replies to requests for information, such as status or help.

x2z Connections - Replies referring to the control and data connections.

x3z Authentication and accounting - Replies for the login process and accounting procedures.

x4z Unspecified as yet.

x5z File system - These replies indicate the status of the Server file system vis-a-vis the requested transfer or other file system action.

The third digit gives a finer gradation of meaning in each of the function categories, specified by the second digit. The list of replies below will illustrate this. Note that the text

associated with each reply is recommended, rather than mandatory, and may even change according to the command with which it is associated. The reply codes, on the other hand, must strictly follow the specifications in the last section; that is, Server implementations should not invent new codes for situations that are only slightly different from the ones described here, but rather should adapt codes already defined.

A command such as TYPE or ALLO whose successful execution does not offer the user-process any new information will cause a 200 reply to be returned. If the command is not implemented by a particular Server-FTP process because it has no relevance to that computer system, for example ALLO at a TOPS20 site, a Positive Completion reply is still desired so that the simple User-process knows it can proceed with its course of action. A 202 reply is used in this case with, for example, the reply text: "No storage allocation necessary." If, on the other hand, the command requests a non-site-specific action and is unimplemented, the response is 502. A refinement of that is the 504 reply for a command that is implemented, but that requests an unimplemented parameter.

4.2.1 Reply Codes by Function Groups

200 Command okay.
500 Syntax error, command unrecognized.
    This may include errors such as command line too long.
501 Syntax error in parameters or arguments.
202 Command not implemented, superfluous at this site.
502 Command not implemented.
503 Bad sequence of commands.
504 Command not implemented for that parameter.

110 Restart marker reply.
    In this case, the text is exact and not left to the particular implementation; it must read:
         MARK yyyy = mmmm
    Where yyyy is User-process data stream marker, and mmmm server's equivalent marker (note the spaces between markers and "=").
211 System status, or system help reply.
212 Directory status.
213 File status.
214 Help message.
    On how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.
215 NAME system type.
    Where NAME is an official system name from the list in the Assigned Numbers document.

120 Service ready in nnn minutes.
220 Service ready for new user.
221 Service closing control connection.
    Logged out if appropriate.
421 Service not available, closing control connection.
    This may be a reply to any command if the service knows it must shut down.
125 Data connection already open; transfer starting.
225 Data connection open; no transfer in progress.
425 Can't open data connection.
226 Closing data connection.
    Requested file action successful (for example, file transfer or file abort).
426 Connection closed; transfer aborted.
227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).

230 User logged in, proceed.
530 Not logged in.
331 User name okay, need password.
332 Need account for login.
532 Need account for storing files.

150 File status okay; about to open data connection.
250 Requested file action okay, completed.
257 "PATHNAME" created.
350 Requested file action pending further information.
450 Requested file action not taken.
    File unavailable (e.g., file busy).
550 Requested action not taken.
    File unavailable (e.g., file not found, no access).
451 Requested action aborted. Local error in processing.
551 Requested action aborted. Page type unknown.
452 Requested action not taken.
    Insufficient storage space in system.
552 Requested file action aborted.
    Exceeded storage allocation (for current directory or
    dataset).
553 Requested action not taken.
    File name not allowed.

4.2.2 Numeric Order List of Reply Codes

110 Restart marker reply.
    In this case, the text is exact and not left to the
    particular implementation; it must read:
         MARK yyyy = mmmm
    Where yyyy is User-process data stream marker, and mmmm
    server's equivalent marker (note the spaces between markers
    and "=").
120 Service ready in nnn minutes.
125 Data connection already open; transfer starting.
150 File status okay; about to open data connection.

200 Command okay.
202 Command not implemented, superfluous at this site.
211 System status, or system help reply.
212 Directory status.
213 File status.
214 Help message.
    On how to use the server or the meaning of a particular
    non-standard command. This reply is useful only to the
    human user.
215 NAME system type.
    Where NAME is an official system name from the list in the
    Assigned Numbers document.
220 Service ready for new user.
221 Service closing control connection.
    Logged out if appropriate.
225 Data connection open; no transfer in progress.
226 Closing data connection.
    Requested file action successful (for example, file
    transfer or file abort).
227 Entering Passive Mode (h1,h2,h3,h4,p1,p2).
230 User logged in, proceed.
250 Requested file action okay, completed.
257 "PATHNAME" created.

331 User name okay, need password.
332 Need account for login.
350 Requested file action pending further information.

421 Service not available, closing control connection.
    This may be a reply to any command if the service knows it
    must shut down.
425 Can't open data connection.
426 Connection closed; transfer aborted.
450 Requested file action not taken.
    File unavailable (e.g., file busy).
451 Requested action aborted: local error in processing.
452 Requested action not taken.
    Insufficient storage space in system.

```
    500 Syntax error, command unrecognized.
        This may include errors such as command line too long.
    501 Syntax error in parameters or arguments.
    502 Command not implemented.
    503 Bad sequence of commands.
    504 Command not implemented for that parameter.
    530 Not logged in.
    532 Need account for storing files.
    550 Requested action not taken.
        File unavailable (e.g., file not found, no access).
    551 Requested action aborted: page type unknown.
    552 Requested file action aborted.
        Exceeded storage allocation (for current directory or
        dataset).
    553 Requested action not taken.
        File name not allowed.
```

5. DECLARATIVE SPECIFICATIONS

5.1. MINIMUM IMPLEMENTATION

In order to make FTP workable without needless error messages, the following minimum implementation is required for all servers:

```
TYPE - ASCII Non-print
MODE - Stream
STRUCTURE - File, Record
COMMANDS - USER, QUIT, PORT,
           TYPE, MODE, STRU,
             for the default values
           RETR, STOR,
           NOOP.
```

The default values for transfer parameters are:

```
TYPE - ASCII Non-print
MODE - Stream
STRU - File
```

All hosts must accept the above as the standard defaults.

5.2. CONNECTIONS

The server protocol interpreter shall "listen" on Port L. The user or user protocol interpreter shall initiate the full-duplex control connection. Server- and user- processes should follow the conventions of the Telnet protocol as specified in the ARPA-Internet Protocol Handbook [1]. Servers are under no obligation to provide for editing of command lines and may require that it be done in the user host. The control connection shall be closed by the server at the user's request after all transfers and replies are completed.

The user-DTP must "listen" on the specified data port; this may be the default user port (U) or a port specified in the PORT command. The server shall initiate the data connection from his own default data port (L-1) using the specified user data port. The direction of the transfer and the port used will be determined by the FTP service command.

Note that all FTP implementation must support data transfer using the default port, and that only the USER-PI may initiate the use of non-default ports.

When data is to be transferred between two servers, A and B (refer to Figure 2), the user-PI, C, sets up control connections with both server-PI's. One of the servers, say A, is then sent a PASV command telling him to "listen" on his data port rather than initiate a connection when he receives a transfer service command. When the user-PI receives an acknowledgment to the PASV command, which includes the identity of the host and port being listened on, the user-PI then sends A's port, a, to B in a PORT command; a reply is returned. The user-PI may then send the corresponding service commands to A and B. Server B initiates the connection and the transfer proceeds. The command-reply sequence is listed below where the messages are vertically synchronous but horizontally asynchronous:

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

C->A : Connect                    C->B : Connect
C->A : PASV
A->C : 227 Entering Passive Mode. A1,A2,A3,A4,a1,a2
                                  C->B : PORT A1,A2,A3,A4,a1,a2
                                  B->C : 200 Okay
C->A : STOR                       C->B : RETR
          B->A : Connect to HOST-A, PORT-a
```

Figure 3

The data connection shall be closed by the server under the conditions described in the Section on Establishing Data Connections. If the data connection is to be closed following a data transfer where closing the connection is not required to indicate the end-of-file, the server must do so immediately. Waiting until after a new transfer command is not permitted because the user-process will have already tested the data connection to see if it needs to do a "listen"; (remember that the user must "listen" on a closed data port BEFORE sending the transfer request). To prevent a race condition here, the server sends a reply (226) after closing the data connection (or if the connection is left open, a "file transfer completed" reply (250) and the user-PI should wait for one of these replies before issuing a new transfer command).

Any time either the user or server see that the connection is being closed by the other side, it should promptly read any remaining data queued on the connection and issue the close on its own side.

5.3. COMMANDS

The commands are Telnet character strings transmitted over the control connections as described in the Section on FTP Commands. The command functions and semantics are described in the Section on Access Control Commands, Transfer Parameter Commands, FTP Service Commands, and Miscellaneous Commands. The command syntax is specified here.

The commands begin with a command code followed by an argument field. The command codes are four or fewer alphabetic characters. Upper and lower case alphabetic characters are to be treated identically. Thus, any of the following may represent the retrieve command:

RETR Retr retr ReTr rETr

This also applies to any symbols representing parameter values, such as A or a for ASCII TYPE. The command codes and the argument fields are separated by one or more spaces.

The argument field consists of a variable length character string ending with the character sequence <CRLF> (Carriage Return, Line Feed) for NVT-ASCII representation; for other negotiated languages a different end of line character might be used. It should be noted that the server is to take no action until the end of line code is received.

The syntax is specified below in NVT-ASCII. All characters in the argument field are ASCII characters including any ASCII represented decimal integers. Square brackets denote an optional argument field. If the option is not taken, the appropriate default is implied.

5.3.1. FTP COMMANDS

The following are the FTP commands:

```
USER <SP> <username> <CRLF>
PASS <SP> <password> <CRLF>
ACCT <SP> <account-information> <CRLF>
CWD  <SP> <pathname> <CRLF>
CDUP <CRLF>
SMNT <SP> <pathname> <CRLF>
QUIT <CRLF>
REIN <CRLF>
PORT <SP> <host-port> <CRLF>
PASV <CRLF>
TYPE <SP> <type-code> <CRLF>
STRU <SP> <structure-code> <CRLF>
MODE <SP> <mode-code> <CRLF>
RETR <SP> <pathname> <CRLF>
STOR <SP> <pathname> <CRLF>
STOU <CRLF>
APPE <SP> <pathname> <CRLF>
ALLO <SP> <decimal-integer>
    [<SP> R <SP> <decimal-integer>] <CRLF>
REST <SP> <marker> <CRLF>
RNFR <SP> <pathname> <CRLF>
RNTO <SP> <pathname> <CRLF>
ABOR <CRLF>
DELE <SP> <pathname> <CRLF>
RMD  <SP> <pathname> <CRLF>
MKD  <SP> <pathname> <CRLF>
PWD  <CRLF>
LIST [<SP> <pathname>] <CRLF>
NLST [<SP> <pathname>] <CRLF>
SITE <SP> <string> <CRLF>
SYST <CRLF>
STAT [<SP> <pathname>] <CRLF>
HELP [<SP> <string>] <CRLF>
NOOP <CRLF>
```


5.3.2. FTP COMMAND ARGUMENTS

The syntax of the above argument fields (using BNF notation where applicable) is:

```
<username> ::= <string>
<password> ::= <string>
<account-information> ::= <string>
<string> ::= <char> | <char><string>
<char> ::= any of the 128 ASCII characters except <CR> and
<LF>
<marker> ::= <pr-string>
<pr-string> ::= <pr-char> | <pr-char><pr-string>
<pr-char> ::= printable characters, any
              ASCII code 33 through 126
<byte-size> ::= <number>
<host-port> ::= <host-number>,<port-number>
<host-number> ::= <number>,<number>,<number>,<number>
<port-number> ::= <number>,<number>
<number> ::= any decimal integer 1 through 255
<form-code> ::= N | T | C
<type-code> ::= A [<sp> <form-code>]
              | E [<sp> <form-code>]
              | I
              | L <sp> <byte-size>
<structure-code> ::= F | R | P
<mode-code> ::= S | B | C
<pathname> ::= <string>
<decimal-integer> ::= any decimal integer
```

5.4. SEQUENCING OF COMMANDS AND REPLIES

The communication between the user and server is intended to be an alternating dialogue. As such, the user issues an FTP command and the server responds with a prompt primary reply. The user should wait for this initial primary success or failure response before sending further commands.

Certain commands require a second reply for which the user should also wait. These replies may, for example, report on the progress or completion of file transfer or the closing of the data connection. They are secondary replies to file transfer commands.

One important group of informational replies is the connection greetings. Under normal circumstances, a server will send a 220 reply, "awaiting input", when the connection is completed. The user should wait for this greeting message before sending any commands. If the server is unable to accept input right away, a 120 "expected delay" reply should be sent immediately and a 220 reply when ready. The user will then know not to hang up if there is a delay.

Spontaneous Replies

Sometimes "the system" spontaneously has a message to be sent to a user (usually all users). For example, "System going down in 15 minutes". There is no provision in FTP for such spontaneous information to be sent from the server to the user. It is recommended that such information be queued in the server-PI and delivered to the user-PI in the next reply (possibly making it a multi-line reply).

The table below lists alternative success and failure replies for each command. These must be strictly adhered to; a server may substitute text in the replies, but the meaning and action implied by the code numbers and by the specific command reply sequence cannot be altered.

Command-Reply Sequences

In this section, the command-reply sequence is presented. Each command is listed with its possible replies; command groups are listed together. Preliminary replies are listed first (with their succeeding replies indented and under them), then positive and negative completion, and finally intermediary

replies with the remaining commands from the sequence following. This listing forms the basis for the state diagrams, which will be presented separately.

```
Connection Establishment
   120
      220
   220
   421
Login
   USER
      230
      530
      500, 501, 421
      331, 332
   PASS
      230
      202
      530
      500, 501, 503, 421
      332
   ACCT
      230
      202
      530
      500, 501, 503, 421
   CWD
      250
      500, 501, 502, 421, 530, 550
   CDUP
      200
      500, 501, 502, 421, 530, 550
   SMNT
      202, 250
      500, 501, 502, 421, 530, 550
Logout
   REIN
      120
         220
      220
      421
      500, 502
   QUIT
      221
      500
```

```
Transfer parameters
   PORT
      200
      500, 501, 421, 530
   PASV
      227
      500, 501, 502, 421, 530
   MODE
      200
      500, 501, 504, 421, 530
   TYPE
      200
      500, 501, 504, 421, 530
   STRU
      200
      500, 501, 504, 421, 530
File action commands
   ALLO
      200
      202
      500, 501, 504, 421, 530
   REST
      500, 501, 502, 421, 530
      350
   STOR
      125, 150
         (110)
         226, 250
         425, 426, 451, 551, 552
      532, 450, 452, 553
      500, 501, 421, 530
   STOU
      125, 150
         (110)
         226, 250
         425, 426, 451, 551, 552
      532, 450, 452, 553
      500, 501, 421, 530
   RETR
      125, 150
         (110)
         226, 250
         425, 426, 451
      450, 550
      500, 501, 421, 530
```



```
LIST
   125, 150
      226, 250
      425, 426, 451
   450
   500, 501, 502, 421, 530
NLST
   125, 150
      226, 250
      425, 426, 451
   450
   500, 501, 502, 421, 530
APPE
   125, 150
      (110)
      226, 250
      425, 426, 451, 551, 552
   532, 450, 550, 452, 553
   500, 501, 502, 421, 530
RNFR
   450, 550
   500, 501, 502, 421, 530
   350
RNTO
   250
   532, 553
   500, 501, 502, 503, 421, 530
DELE
   250
   450, 550
   500, 501, 502, 421, 530
RMD
   250
   500, 501, 502, 421, 530, 550
MKD
   257
   500, 501, 502, 421, 530, 550
PWD
   257
   500, 501, 502, 421, 550
ABOR
   225, 226
   500, 501, 502, 421
```

```
Informational commands
   SYST
      215
      500, 501, 502, 421
   STAT
      211, 212, 213
      450
      500, 501, 502, 421, 530
   HELP
      211, 214
      500, 501, 502, 421
Miscellaneous commands
   SITE
      200
      202
      500, 501, 530
   NOOP
      200
      500 421
```

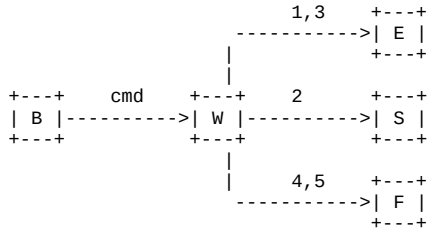

6. STATE DIAGRAMS

Here we present state diagrams for a very simple minded FTP implementation. Only the first digit of the reply codes is used. There is one state diagram for each group of FTP commands or command sequences.

The command groupings were determined by constructing a model for each command then collecting together the commands with structurally identical models.

For each command or command sequence there are three possible outcomes: success (S), failure (F), and error (E). In the state diagrams below we use the symbol B for "begin", and the symbol W for "wait for reply".

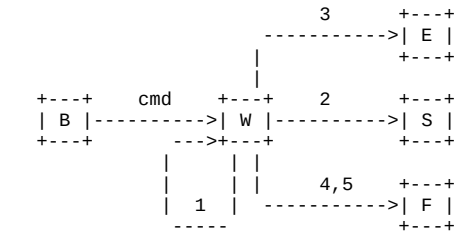
We first present the diagram that represents the largest group of FTP commands:



This diagram models the commands:

ABOR, ALLO, DELE, CWD, CDUP, SMNT, HELP, MODE, NOOP, PASV, QUIT, SITE, PORT, SYST, STAT, RMD, MKD, PWD, STRU, and TYPE.

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

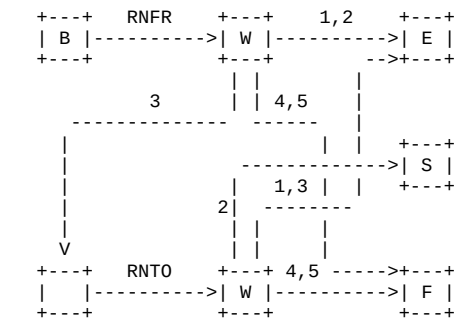


This diagram models the commands:

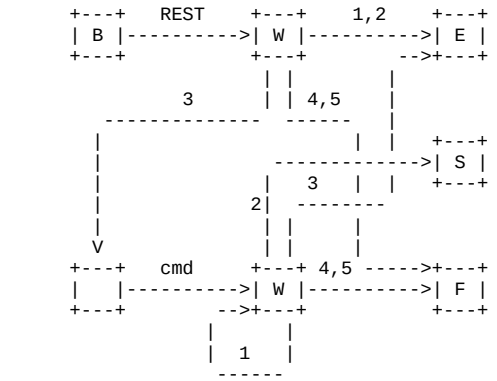
APPE, LIST, NLST, REIN, RETR, STOR, and STOU.

Note that this second model could also be used to represent the first group of commands, the only difference being that in the first group the 100 series replies are unexpected and therefore treated as error, while the second group expects (some may require) 100 series replies. Remember that at most, one 100 series reply is allowed per command.

The remaining diagrams model command sequences, perhaps the simplest of these is the rename sequence:



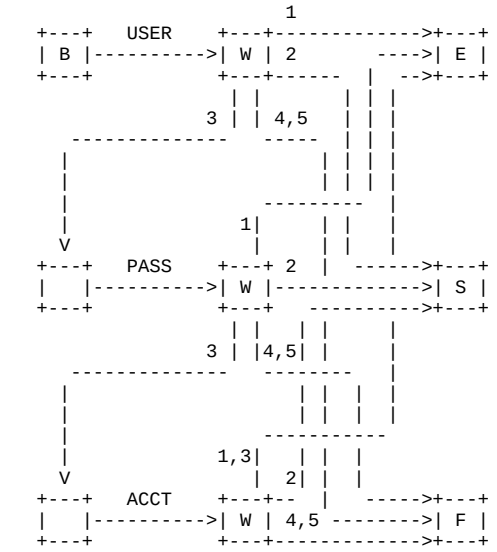
The next diagram is a simple model of the Restart command:



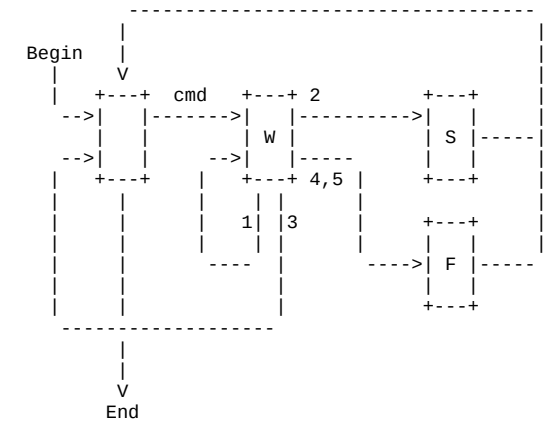
Where "cmd" is APPE, STOR, or RETR.

We note that the above three models are similar. The Restart differs from the Rename two only in the treatment of 100 series replies at the second stage, while the second group expects (some may require) 100 series replies. Remember that at most, one 100 series reply is allowed per command.

The most complicated diagram is for the Login sequence:



Finally, we present a generalized diagram that could be used to model the command and reply interchange:



7. TYPICAL FTP SCENARIO

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

In general, the user will communicate to the server via a mediating user-FTP process. The following may be a typical scenario. The user-FTP prompts are shown in parentheses, '---->' represents commands from host U to host S, and '<----' represents replies from host S to host U.

```
   LOCAL COMMANDS BY USER              ACTION INVOLVED

   ftp (host) multics<CR>          Connect to host S, port L,
                                   establishing control connections.
                                   <---- 220 Service ready <CRLF>.
   username Doe <CR>               USER Doe<CRLF>---->
                                   <---- 331 User name ok,
                                              need password<CRLF>.
   password mumble <CR>            PASS mumble<CRLF>---->
                                   <---- 230 User logged in<CRLF>.
   retrieve (local type) ASCII<CR>
   (local pathname) test 1 <CR>    User-FTP opens local file in ASCII.
   (for. pathname) test.pl1<CR>    RETR test.pl1<CRLF> ---->
                                   <---- 150 File status okay;
                                         about to open data
                                         connection<CRLF>.
                                   Server makes data connection
                                   to port U.

                                   <---- 226 Closing data connection,
                                       file transfer successful<CRLF>.
   type Image<CR>                  TYPE I<CRLF> ---->
                                   <---- 200 Command OK<CRLF>
   store (local type) image<CR>
   (local pathname) file dump<CR>  User-FTP opens local file in Image.
   (for.pathname) >udd>cn>fd<CR>   STOR >udd>cn>fd<CRLF> ---->
                                   <---- 550 Access denied<CRLF>
   terminate                       QUIT <CRLF> ---->
                                   Server closes all
                                   connections.
```

8. CONNECTION ESTABLISHMENT

The FTP control connection is established via TCP between the user process port U and the server process port L. This protocol is assigned the service port 21 (25 octal), that is L=21.

APPENDIX I - PAGE STRUCTURE

The need for FTP to support page structure derives principally from the need to support efficient transmission of files between TOPS-20 systems, particularly the files used by NLS.

The file system of TOPS-20 is based on the concept of pages. The operating system is most efficient at manipulating files as pages. The operating system provides an interface to the file system so that many applications view files as sequential streams of characters. However, a few applications use the underlying page structures directly, and some of these create holey files.

A TOPS-20 disk file consists of four things: a pathname, a page table, a (possibly empty) set of pages, and a set of attributes.

The pathname is specified in the RETR or STOR command. It includes the directory name, file name, file name extension, and generation number.

The page table contains up to 2**18 entries. Each entry may be EMPTY, or may point to a page. If it is not empty, there are also some page-specific access bits; not all pages of a file need have the same access protection.

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

The attributes of the file, in the File Descriptor Block (FDB), contain such things as creation time, write time, read time, writer's byte-size, end-of-file pointer, count of reads and writes, backup system tape numbers, etc.

Note that there is NO requirement that entries in the page table be contiguous. There may be empty page table slots between occupied ones. Also, the end of file pointer is simply a number. There is no requirement that it in fact point at the "last" datum in the file. Ordinary sequential I/O calls in TOPS-20 will cause the end of file pointer to be left after the last datum written, but other operations may cause it not to be so, if a particular programming system so requires.

In fact, in both of these special cases, "holey" files and end-of-file pointers NOT at the end of the file, occur with NLS data files.

The TOPS-20 paged files can be sent with the FTP transfer parameters: TYPE L 36, STRU P, and MODE S (in fact, any mode could be used).

Each page of information has a header. Each header field, which is a logical byte, is a TOPS-20 word, since the TYPE is L 36.

The header fields are:

Word 0: Header Length.

The header length is 5.

Word 1: Page Index.

If the data is a disk file page, this is the number of that page in the file's page map. Empty pages (holes) in the file are simply not sent. Note that a hole is NOT the same as a page of zeros.

Word 2: Data Length.

The number of data words in this page, following the header. Thus, the total length of the transmission unit is the Header Length plus the Data Length.

Word 3: Page Type.

A code for what type of chunk this is. A data page is type 3, the FDB page is type 2.

Word 4: Page Access Control.

The access bits associated with the page in the file's page map. (This full word quantity is put into AC2 of an SPACS by the program reading from net to disk.)

After the header are Data Length data words. Data Length is currently either 512 for a data page or 31 for an FDB. Trailing zeros in a disk file page may be discarded, making Data Length less than 512 in that case.

APPENDIX II - DIRECTORY COMMANDS

Since UNIX has a tree-like directory structure in which directories are as easy to manipulate as ordinary files, it is useful to expand the FTP servers on these machines to include commands which deal with the creation of directories. Since there are other hosts on the ARPA-Internet which have tree-like directories (including TOPS-20 and Multics), these commands are as general as possible.

Four directory commands have been added to FTP:

MKD pathname

Make a directory with the name "pathname".

RMD pathname

Remove the directory with the name "pathname".

PWD

Print the current working directory name.

CDUP

Change to the parent of the current working directory.

The "pathname" argument should be created (removed) as a subdirectory of the current working directory, unless the "pathname" string contains sufficient information to specify otherwise to the server, e.g., "pathname" is an absolute pathname (in UNIX and Multics), or pathname is something like "<abso.lute.path>" to TOPS-20.

REPLY CODES

The CDUP command is a special case of CWD, and is included to simplify the implementation of programs for transferring directory trees between operating systems having different syntaxes for naming the parent directory. The reply codes for CDUP be identical to the reply codes of CWD.

The reply codes for RMD be identical to the reply codes for its file analogue, DELE.

The reply codes for MKD, however, are a bit more complicated. A freshly created directory will probably be the object of a future

CWD command. Unfortunately, the argument to MKD may not always be a suitable argument for CWD. This is the case, for example, when a TOPS-20 subdirectory is created by giving just the subdirectory name. That is, with a TOPS-20 server FTP, the command sequence

```
MKD MYDIR
CWD MYDIR
```

will fail. The new directory may only be referred to by its "absolute" name; e.g., if the MKD command above were issued while connected to the directory <DFRANKLIN>, the new subdirectory could only be referred to by the name <DFRANKLIN.MYDIR>.

Even on UNIX and Multics, however, the argument given to MKD may not be suitable. If it is a "relative" pathname (i.e., a pathname which is interpreted relative to the current directory), the user would need to be in the same current directory in order to reach the subdirectory. Depending on the application, this may be inconvenient. It is not very robust in any case.

To solve these problems, upon successful completion of an MKD command, the server should return a line of the form:

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

That is, the server will tell the user what string to use when referring to the created directory. The directory name can contain any character; embedded double-quotes should be escaped by double-quotes (the "quote-doubling" convention).

For example, a user connects to the directory /usr/dm, and creates a subdirectory, named pathname:

```
CWD /usr/dm
200 directory changed to /usr/dm
MKD pathname
257 "/usr/dm/pathname" directory created
```

An example with an embedded double quote:

```
MKD foo"bar
257 "/usr/dm/foo""bar" directory created
CWD /usr/dm/foo"bar
200 directory changed to /usr/dm/foo"bar
```


The prior existence of a subdirectory with the same name is an error, and the server must return an "access denied" error reply in that case.

```
CWD /usr/dm
200 directory changed to /usr/dm
MKD pathname
521-"/usr/dm/pathname" directory already exists;
521 taking no action.
```

The failure replies for MKD are analogous to its file creating cousin, STOR. Also, an "access denied" return is given if a file name with the same name as the subdirectory will conflict with the creation of the subdirectory (this is a problem on UNIX, but shouldn't be one on TOPS-20).

Essentially because the PWD command returns the same type of information as the successful MKD command, the successful PWD command uses the 257 reply code as well.

SUBTLETIES

Because these commands will be most useful in transferring subtrees from one machine to another, carefully observe that the argument to MKD is to be interpreted as a sub-directory of the current working directory, unless it contains enough information for the destination host to tell otherwise. A hypothetical example of its use in the TOPS-20 world:

```
CWD <some.where>
200 Working directory changed
MKD overrainbow
257 "<some.where.overrainbow>" directory created
CWD overrainbow
431 No such directory
CWD <some.where.overrainbow>
200 Working directory changed

CWD <some.where>
200 Working directory changed to <some.where>
MKD <unambiguous>
257 "<unambiguous>" directory created
CWD <unambiguous>
```

Note that the first example results in a subdirectory of the connected directory. In contrast, the argument in the second example contains enough information for TOPS-20 to tell that the

<unambiguous> directory is a top-level directory. Note also that in the first example the user "violated" the protocol by attempting to access the freshly created directory with a name other than the one returned by TOPS-20. Problems could have resulted in this case had there been an <overrainbow> directory; this is an ambiguity inherent in some TOPS-20 implementations. Similar considerations apply to the RMD command. The point is this: except where to do so would violate a host's conventions for denoting relative versus absolute pathnames, the host should treat the operands of the MKD and RMD commands as subdirectories. The 257 reply to the MKD command must always contain the absolute pathname of the created directory.

APPENDIX III - RFCs on FTP

Bhushan, Abhay, "A File Transfer Protocol", RFC 114 (NIC 5823), MIT-Project MAC, 16 April 1971.

Harslem, Eric, and John Heafner, "Comments on RFC 114 (A File Transfer Protocol)", RFC 141 (NIC 6726), RAND, 29 April 1971.

Bhushan, Abhay, et al, "The File Transfer Protocol", RFC 172 (NIC 6794), MIT-Project MAC, 23 June 1971.

Braden, Bob, "Comments on DTP and FTP Proposals", RFC 238 (NIC 7663), UCLA/CCN, 29 September 1971.

Bhushan, Abhay, et al, "The File Transfer Protocol", RFC 265 (NIC 7813), MIT-Project MAC, 17 November 1971.

McKenzie, Alex, "A Suggested Addition to File Transfer Protocol", RFC 281 (NIC 8163), BBN, 8 December 1971.

Bhushan, Abhay, "The Use of "Set Data Type" Transaction in File Transfer Protocol", RFC 294 (NIC 8304), MIT-Project MAC, 25 January 1972.

Bhushan, Abhay, "The File Transfer Protocol", RFC 354 (NIC 10596), MIT-Project MAC, 8 July 1972.

Bhushan, Abhay, "Comments on the File Transfer Protocol (RFC 354)", RFC 385 (NIC 11357), MIT-Project MAC, 18 August 1972.

Hicks, Greg, "User FTP Documentation", RFC 412 (NIC 12404), Utah, 27 November 1972.

Bhushan, Abhay, "File Transfer Protocol (FTP) Status and Further Comments", RFC 414 (NIC 12406), MIT-Project MAC, 20 November 1972.

Braden, Bob, "Comments on File Transfer Protocol", RFC 430 (NIC 13299), UCLA/CCN, 7 February 1973.

Thomas, Bob, and Bob Clements, "FTP Server-Server Interaction", RFC 438 (NIC 13770), BBN, 15 January 1973.

Braden, Bob, "Print Files in FTP", RFC 448 (NIC 13299), UCLA/CCN, 27 February 1973.

McKenzie, Alex, "File Transfer Protocol", RFC 454 (NIC 14333), BBN, 16 February 1973.

Bressler, Bob, and Bob Thomas, "Mail Retrieval via FTP", RFC 458 (NIC 14378), BBN-NET and BBN-TENEX, 20 February 1973.

Neigus, Nancy, "File Transfer Protocol", RFC 542 (NIC 17759), BBN, 12 July 1973.

Krilanovich, Mark, and George Gregg, "Comments on the File Transfer Protocol", RFC 607 (NIC 21255), UCSB, 7 January 1974.

Pogran, Ken, and Nancy Neigus, "Response to RFC 607 - Comments on the File Transfer Protocol", RFC 614 (NIC 21530), BBN, 28 January 1974.

Krilanovich, Mark, George Gregg, Wayne Hathaway, and Jim White, "Comments on the File Transfer Protocol", RFC 624 (NIC 22054), UCSB, Ames Research Center, SRI-ARC, 28 February 1974.

Bhushan, Abhay, "FTP Comments and Response to RFC 430", RFC 463 (NIC 14573), MIT-DMCG, 21 February 1973.

Braden, Bob, "FTP Data Compression", RFC 468 (NIC 14742), UCLA/CCN, 8 March 1973.

Bhushan, Abhay, "FTP and Network Mail System", RFC 475 (NIC 14919), MIT-DMCG, 6 March 1973.

Bressler, Bob, and Bob Thomas "FTP Server-Server Interaction - II", RFC 478 (NIC 14947), BBN-NET and BBN-TENEX, 26 March 1973.

White, Jim, "Use of FTP by the NIC Journal", RFC 479 (NIC 14948), SRI-ARC, 8 March 1973.

White, Jim, "Host-Dependent FTP Parameters", RFC 480 (NIC 14949), SRI-ARC, 8 March 1973.

Padlipsky, Mike, "An FTP Command-Naming Problem", RFC 506 (NIC 16157), MIT-Multics, 26 June 1973.

Day, John, "Memo to FTP Group (Proposal for File Access Protocol)", RFC 520 (NIC 16819), Illinois, 25 June 1973.

Merryman, Robert, "The UCSD-CC Server-FTP Facility", RFC 532 (NIC 17451), UCSD-CC, 22 June 1973.

Braden, Bob, "TENEX FTP Problem", RFC 571 (NIC 18974), UCLA/CCN, 15 November 1973.

McKenzie, Alex, and Jon Postel, "Telnet and FTP Implementation - Schedule Change", RFC 593 (NIC 20615), BBN and MITRE, 29 November 1973.

Sussman, Julie, "FTP Error Code Usage for More Reliable Mail Service", RFC 630 (NIC 30237), BBN, 10 April 1974.

Postel, Jon, "Revised FTP Reply Codes", RFC 640 (NIC 30843), UCLA/NMC, 5 June 1974.

Harvey, Brian, "Leaving Well Enough Alone", RFC 686 (NIC 32481), SU-AI, 10 May 1975.

Harvey, Brian, "One More Try on the FTP", RFC 691 (NIC 32700), SU-AI, 28 May 1975.

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

Harrenstien, Ken, "FTP Extension: XSEN", RFC 737 (NIC 42217), SRI-KL, 31 October 1977.

Harrenstien, Ken, "FTP Extension: XRSQ/XRCP", RFC 743 (NIC 42758), SRI-KL, 30 December 1977.

Lebling, P. David, "Survey of FTP Mail and MLFL", RFC 751, MIT, 10 December 1978.

Postel, Jon, "File Transfer Protocol Specification", RFC 765, ISI, June 1980.

Mankins, David, Dan Franklin, and Buzz Owen, "Directory Oriented FTP Commands", RFC 776, BBN, December 1980.

Padlipsky, Michael, "FTP Unique-Named Store Command", RFC 949, MITRE, July 1985.

REFERENCES

[1] Feinler, Elizabeth, "Internet Protocol Transition Workbook", Network Information Center, SRI International, March 1982.

[2] Postel, Jon, "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, DARPA, September 1981.

[3] Postel, Jon, and Joyce Reynolds, "Telnet Protocol Specification", RFC 854, ISI, May 1983.

[4] Reynolds, Joyce, and Jon Postel, "Assigned Numbers", RFC 943, ISI, April 1985.

Network Working Group D. Kristol
Request for Comments: 2109 Bell Laboratories, Lucent Technologies
Category: Standards Track L. Montulli
Netscape Communications
February 1997

HTTP State Management Mechanism

Status of this Memo

1. ABSTRACT

This document specifies a way to create a stateful session with HTTP requests and responses. It describes two new headers, Cookie and Set-Cookie, which carry state information between participating origin servers and user agents. The method described here differs from Netscape's Cookie proposal, but it can interoperate with HTTP/1.0 user agents that use Netscape's method. (See the HISTORICAL section.)

2. TERMINOLOGY

The terms user agent, client, server, proxy, and origin server have the same meaning as in the HTTP/1.0 specification.

Fully-qualified host name (FQHN) means either the fully-qualified domain name (FQDN) of a host (i.e., a completely specified domain name ending in a top-level domain such as .com or .uk), or the numeric Internet Protocol (IP) address of a host. The fully qualified domain name is preferred; use of numeric IP addresses is strongly discouraged.

The terms request-host and request-URI refer to the values the client would send to the server as, respectively, the host (but not port) and abs_path portions of the absoluteURI (http_URL) of the HTTP request line. Note that request-host must be a FQHN.

Hosts names can be specified either as an IP address or a FQHN string. Sometimes we compare one host name with another. Host A's name domain-matches host B's if

* both host names are IP addresses and their host name strings match exactly; or

* both host names are FQDN strings and their host name strings match exactly; or

* A is a FQDN string and has the form NB, where N is a non-empty name string, B has the form .B', and B' is a FQDN string. (So, x.y.com domain-matches .y.com but not y.com.)

Note that domain-match is not a commutative operation: a.b.c.com domain-matches .c.com, but not the reverse.

Because it was used in Netscape's original implementation of state management, we will use the term cookie to refer to the state information that passes between an origin server and user agent, and that gets stored by the user agent.

3. STATE AND SESSIONS

This document describes a way to create stateful sessions with HTTP requests and responses. Currently, HTTP servers respond to each client request without relating that request to previous or subsequent requests; the technique allows clients and servers that wish to exchange state information to place HTTP requests and responses within a larger context, which we term a "session". This context might be used to create, for example, a "shopping cart", in which user selections can be aggregated before purchase, or a magazine browsing system, in which a user's previous reading affects which offerings are presented.

There are, of course, many different potential contexts and thus many different potential types of session. The designers' paradigm for sessions created by the exchange of cookies has these key attributes:

1. Each session has a beginning and an end.

2. Each session is relatively short-lived.

3. Either the user agent or the origin server may terminate a session.

4. The session is implicit in the exchange of state information.

4. OUTLINE

We outline here a way for an origin server to send state information to the user agent, and for the user agent to return the state information to the origin server. The goal is to have a minimal impact on HTTP and user agents. Only origin servers that need to maintain sessions would suffer any significant impact, and that impact can largely be confined to Common Gateway Interface (CGI) programs, unless the server provides more sophisticated state management support. (See Implementation Considerations, below.)

4.1 Syntax: General

The two state management headers, Set-Cookie and Cookie, have common syntactic properties involving attribute-value pairs. The following grammar uses the notation, and tokens DIGIT (decimal digits) and token (informally, a sequence of non-special, non-white space characters) from the HTTP/1.1 specification [RFC 2068] to describe their syntax.

```
av-pairs        =       av-pair *(";" av-pair)
av-pair         =       attr ["=" value]        ; optional value
attr            =       token
value           =       word
word            =       token | quoted-string
```

Attributes (names) (attr) are case-insensitive. White space is permitted between tokens. Note that while the above syntax description shows value as optional, most attrs require them.

NOTE: The syntax above allows whitespace between the attribute and the = sign.

4.2 Origin Server Role

4.2.1 General

The origin server initiates a session, if it so desires. (Note that "session" here does not refer to a persistent network connection but to a logical session created from HTTP requests and responses. The presence or absence of a persistent connection should have no effect on the use of cookie-derived sessions). To initiate a session, the origin server returns an extra response header to the client, Set-Cookie. (The details follow later.)

A user agent returns a Cookie request header (see below) to the origin server if it chooses to continue a session. The origin server may ignore it or use it to determine the current state of the session. It may send back to the client a Set-Cookie response header with the same or different information, or it may send no Set-Cookie header at all. The origin server effectively ends a session by sending the client a Set-Cookie header with Max-Age=0.

Servers may return a Set-Cookie response headers with any response. User agents should send Cookie request headers, subject to other rules detailed below, with every request.

An origin server may include multiple Set-Cookie headers in a response. Note that an intervening gateway could fold multiple such headers into a single header.

4.2.2 Set-Cookie Syntax

The syntax for the Set-Cookie response header is

```
set-cookie      =       "Set-Cookie:" cookies
cookies         =       1#cookie
cookie          =       NAME "=" VALUE *(";" cookie-av)
NAME            =       attr
VALUE           =       value
cookie-av       =       "Comment" "=" value
                |       "Domain" "=" value
                |       "Max-Age" "=" value
                |       "Path" "=" value
                |       "Secure"
                |       "Version" "=" 1*DIGIT
```

Informally, the Set-Cookie response header comprises the token Set-Cookie:, followed by a comma-separated list of one or more cookies. Each cookie begins with a NAME=VALUE pair, followed by zero or more semi-colon-separated attribute-value pairs. The syntax for attribute-value pairs was shown earlier. The specific attributes and the semantics of their values follows. The NAME=VALUE attribute-value pair must come first in each cookie. The others, if present, can occur in any order. If an attribute appears more than once in a cookie, the behavior is undefined.

NAME=VALUE
: Required. The name of the state information ("cookie") is NAME, and its value is VALUE. NAMEs that begin with $ are reserved for other uses and must not be used by applications.

The VALUE is opaque to the user agent and may be anything the origin server chooses to send, possibly in a server-selected printable ASCII encoding. "Opaque" implies that the content is of interest and relevance only to the origin server. The content may, in fact, be readable by anyone that examines the Set-Cookie header.

Comment=comment
Optional. Because cookies can contain private information about a user, the Cookie attribute allows an origin server to document its intended use of a cookie. The user can inspect the information to decide whether to initiate or continue a session with this cookie.

Domain=domain
Optional. The Domain attribute specifies the domain for which the cookie is valid. An explicitly specified domain must always start with a dot.

Max-Age=delta-seconds
Optional. The Max-Age attribute defines the lifetime of the cookie, in seconds. The delta-seconds value is a decimal non-negative integer. After delta-seconds seconds elapse, the client should discard the cookie. A value of zero means the cookie should be discarded immediately.

Path=path
Optional. The Path attribute specifies the subset of URLs to which this cookie applies.

Secure
Optional. The Secure attribute (with no value) directs the user agent to use only (unspecified) secure means to contact the origin server whenever it sends back this cookie.

The user agent (possibly under the user's control) may determine what level of security it considers appropriate for "secure" cookies. The Secure attribute should be considered security advice from the server to the user agent, indicating that it is in the session's interest to protect the cookie contents.

Version=version
Required. The Version attribute, a decimal integer, identifies to which version of the state management specification the cookie conforms. For this specification, Version=1 applies.

4.2.3 Controlling Caching

An origin server must be cognizant of the effect of possible caching of both the returned resource and the Set-Cookie header. Caching "public" documents is desirable. For example, if the origin server wants to use a public document such as a "front door" page as a sentinel to indicate the beginning of a session for which a Set-Cookie response header must be generated, the page should be stored in caches "pre-expired" so that the origin server will see further requests. "Private documents", for example those that contain information strictly private to a session, should not be cached in shared caches.

If the cookie is intended for use by a single user, the Set-cookie header should not be cached. A Set-cookie header that is intended to be shared by multiple users may be cached.

The origin server should send the following additional HTTP/1.1 response headers, depending on circumstances:

* To suppress caching of the Set-Cookie header: Cache-control: no-cache="set-cookie".

and one of the following:

* To suppress caching of a private document in shared caches: Cache-control: private.

* To allow caching of a document and require that it be validated before returning it to the client: Cache-control: must-revalidate.

* To allow caching of a document, but to require that proxy caches (not user agent caches) validate it before returning it to the client: Cache-control: proxy-revalidate.

* To allow caching of a document and request that it be validated before returning it to the client (by "pre-expiring" it): Cache-control: max-age=0. Not all caches will revalidate the document in every case.

HTTP/1.1 servers must send Expires: old-date (where old-date is a date long in the past) on responses containing Set-Cookie response headers unless they know for certain (by out of band means) that there are no downsteam HTTP/1.0 proxies. HTTP/1.1 servers may send other Cache-Control directives that permit caching by HTTP/1.1 proxies in addition to the Expires: old-date directive; the Cache-Control directive will override the Expires: old-date for HTTP/1.1 proxies.

4.3 User Agent Role

4.3.1 Interpreting Set-Cookie

The user agent keeps separate track of state information that arrives via Set-Cookie response headers from each origin server (as distinguished by name or IP address and port). The user agent applies these defaults for optional attributes that are missing:

Version Defaults to "old cookie" behavior as originally specified by Netscape. See the HISTORICAL section.

Domain Defaults to the request-host. (Note that there is no dot at the beginning of request-host.)

Max-Age The default behavior is to discard the cookie when the user agent exits.

Path Defaults to the path of the request URL that generated the Set-Cookie response, up to, but not including, the right-most /.

Secure If absent, the user agent may send the cookie over an insecure channel.

4.3.2 Rejecting Cookies

To prevent possible security or privacy violations, a user agent rejects a cookie (shall not store its information) if any of the following is true:

* The value for the Path attribute is not a prefix of the request-URI.

* The value for the Domain attribute contains no embedded dots or does not start with a dot.

* The value for the request-host does not domain-match the Domain attribute.

* The request-host is a FQDN (not IP address) and has the form HD, where D is the value of the Domain attribute, and H is a string that contains one or more dots.

Examples:

* A Set-Cookie from request-host y.x.foo.com for Domain=.foo.com would be rejected, because H is y.x and contains a dot.

* A Set-Cookie from request-host x.foo.com for Domain=.foo.com would be accepted.

* A Set-Cookie with Domain=.com or Domain=.com., will always be rejected, because there is no embedded dot.

* A Set-Cookie with Domain=ajax.com will be rejected because the value for Domain does not begin with a dot.

4.3.3 Cookie Management

If a user agent receives a Set-Cookie response header whose NAME is the same as a pre-existing cookie, and whose Domain and Path attribute values exactly (string) match those of a pre-existing cookie, the new cookie supersedes the old. However, if the Set-Cookie has a value for Max-Age of zero, the (old and new) cookie is discarded. Otherwise cookies accumulate until they expire (resources permitting), at which time they are discarded.

Because user agents have finite space in which to store cookies, they may also discard older cookies to make space for newer ones, using, for example, a least-recently-used algorithm, along with constraints on the maximum number of cookies that each origin server may set.

If a Set-Cookie response header includes a Comment attribute, the user agent should store that information in a human-readable form with the cookie and should display the comment text as part of a cookie inspection user interface.

User agents should allow the user to control cookie destruction. An infrequently-used cookie may function as a "preferences file" for network applications, and a user may wish to keep it even if it is the least-recently-used cookie. One possible implementation would be an interface that allows the permanent storage of a cookie through a checkbox (or, conversely, its immediate destruction).

Privacy considerations dictate that the user have considerable control over cookie management. The PRIVACY section contains more information.

4.3.4 Sending Cookies to the Origin Server

When it sends a request to an origin server, the user agent sends a Cookie request header to the origin server if it has cookies that are applicable to the request, based on

* the request-host;

* the request-URI;

* the cookie's age.

The syntax for the header is:

```
cookie          =       "Cookie:" cookie-version
                        1*((";" | ",") cookie-value)
cookie-value    =       NAME "=" VALUE [";" path] [";" domain]
cookie-version  =       "$Version" "=" value
NAME            =       attr
VALUE           =       value
path            =       "$Path" "=" value
domain          =       "$Domain" "=" value
```

The value of the cookie-version attribute must be the value from the Version attribute, if any, of the corresponding Set-Cookie response header. Otherwise the value for cookie-version is 0. The value for the path attribute must be the value from the Path attribute, if any, of the corresponding Set-Cookie response header. Otherwise the attribute should be omitted from the Cookie request header. The value for the domain attribute must be the value from the Domain attribute, if any, of the corresponding Set-Cookie response header. Otherwise the attribute should be omitted from the Cookie request header.

Note that there is no Comment attribute in the Cookie request header corresponding to the one in the Set-Cookie response header. The user agent does not return the comment information to the origin server.

The following rules apply to choosing applicable cookie-values from among all the cookies the user agent has.

Domain Selection
: The origin server's fully-qualified host name must domain-match the Domain attribute of the cookie.

Path Selection
: The Path attribute of the cookie must match a prefix of the request-URI.

Max-Age Selection
: Cookies that have expired should have been discarded and thus are not forwarded to an origin server.

If multiple cookies satisfy the criteria above, they are ordered in the Cookie header such that those with more specific Path attributes precede those with less specific. Ordering with respect to other attributes (e.g., Domain) is unspecified.

Note: For backward compatibility, the separator in the Cookie header is semi-colon (;) everywhere. A server should also accept comma (,) as the separator between cookie-values for future compatibility.

4.3.5 Sending Cookies in Unverifiable Transactions

Users must have control over sessions in order to ensure privacy. (See PRIVACY section below.) To simplify implementation and to prevent an additional layer of complexity where adequate safeguards exist, however, this document distinguishes between transactions that are verifiable and those that are unverifiable. A transaction is verifiable if the user has the option to review the request-URI prior to its use in the transaction. A transaction is unverifiable if the user does not have that option. Unverifiable transactions typically arise when a user agent automatically requests inlined or embedded entities or when it resolves redirection (3xx) responses from an origin server. Typically the origin transaction, the transaction that the user initiates, is verifiable, and that transaction may directly or indirectly induce the user agent to make unverifiable transactions.

When it makes an unverifiable transaction, a user agent must enable a session only if a cookie with a domain attribute D was sent or received in its origin transaction, such that the host name in the Request-URI of the unverifiable transaction domain-matches D.

This restriction prevents a malicious service author from using unverifiable transactions to induce a user agent to start or continue a session with a server in a different domain. The starting or continuation of such sessions could be contrary to the privacy expectations of the user, and could also be a security problem.

User agents may offer configurable options that allow the user agent, or any autonomous programs that the user agent executes, to ignore the above rule, so long as these override options default to "off".

Many current user agents already provide a review option that would render many links verifiable. For instance, some user agents display the URL that would be referenced for a particular link when the mouse pointer is placed over that link. The user can therefore determine whether to visit that site before causing the browser to do so. (Though not implemented on current user agents, a similar technique could be used for a button used to submit a form -- the user agent

could display the action to be taken if the user were to select that button.) However, even this would not make all links verifiable; for example, links to automatically loaded images would not normally be subject to "mouse pointer" verification.

Many user agents also provide the option for a user to view the HTML source of a document, or to save the source to an external file where it can be viewed by another application. While such an option does provide a crude review mechanism, some users might not consider it acceptable for this purpose.

4.4 How an Origin Server Interprets the Cookie Header

A user agent returns much of the information in the Set-Cookie header to the origin server when the Path attribute matches that of a new request. When it receives a Cookie header, the origin server should treat cookies with NAMEs whose prefix is $ specially, as an attribute for the adjacent cookie. The value for such a NAME is to be interpreted as applying to the lexically (left-to-right) most recent cookie whose name does not have the $ prefix. If there is no previous cookie, the value applies to the cookie mechanism as a whole. For example, consider the cookie

```
Cookie: $Version="1"; Customer="WILE_E_COYOTE";
        $Path="/acme"
```

$Version applies to the cookie mechanism as a whole (and gives the version number for the cookie mechanism). $Path is an attribute whose value (/acme) defines the Path attribute that was used when the Customer cookie was defined in a Set-Cookie response header.

4.5 Caching Proxy Role

One reason for separating state information from both a URL and document content is to facilitate the scaling that caching permits. To support cookies, a caching proxy must obey these rules already in the HTTP specification:

* Honor requests from the cache, if possible, based on cache validity rules.

* Pass along a Cookie request header in any request that the proxy must make of another server.

* Return the response to the client. Include any Set-Cookie response header.

* Cache the received response subject to the control of the usual headers, such as Expires, Cache-control: no-cache, and Cache-control: private,

* Cache the Set-Cookie subject to the control of the usual header, Cache-control: no-cache="set-cookie". (The Set-Cookie header should usually not be cached.)

Proxies must not introduce Set-Cookie (Cookie) headers of their own in proxy responses (requests).

5. EXAMPLES

5.1 Example 1

Most detail of request and response headers has been omitted. Assume the user agent has no stored cookies.

1. User Agent -> Server

   ```
   POST /acme/login HTTP/1.1
   [form data]
   ```

   User identifies self via a form.

2. Server -> User Agent

   ```
   HTTP/1.1 200 OK
   Set-Cookie: Customer="WILE_E_COYOTE"; Version="1"; Path="/acme"
   ```

   Cookie reflects user's identity.

3. User Agent -> Server

   ```
   POST /acme/pickitem HTTP/1.1
   Cookie: $Version="1"; Customer="WILE_E_COYOTE"; $Path="/acme"
   [form data]
   ```

   User selects an item for "shopping basket."

4. Server -> User Agent

   ```
   HTTP/1.1 200 OK
   Set-Cookie: Part_Number="Rocket_Launcher_0001"; Version="1";
           Path="/acme"
   ```

   Shopping basket contains an item.

5. User Agent -> Server

```
POST /acme/shipping HTTP/1.1
Cookie: $Version="1";
        Customer="WILE_E_COYOTE"; $Path="/acme";
        Part_Number="Rocket_Launcher_0001"; $Path="/acme"
[form data]
```

User selects shipping method from form.

6. Server -> User Agent

```
HTTP/1.1 200 OK
Set-Cookie: Shipping="FedEx"; Version="1"; Path="/acme"
```

New cookie reflects shipping method.

7. User Agent -> Server

```
POST /acme/process HTTP/1.1
Cookie: $Version="1";
        Customer="WILE_E_COYOTE"; $Path="/acme";
        Part_Number="Rocket_Launcher_0001"; $Path="/acme";
        Shipping="FedEx"; $Path="/acme"
[form data]
```

User chooses to process order.

8. Server -> User Agent

```
HTTP/1.1 200 OK
```

Transaction is complete.

The user agent makes a series of requests on the origin server, after each of which it receives a new cookie. All the cookies have the same Path attribute and (default) domain. Because the request URLs all have /acme as a prefix, and that matches the Path attribute, each request contains all the cookies received so far.

5.2 Example 2

This example illustrates the effect of the Path attribute. All detail of request and response headers has been omitted. Assume the user agent has no stored cookies.

Imagine the user agent has received, in response to earlier requests, the response headers


```
Set-Cookie: Part_Number="Rocket_Launcher_0001"; Version="1";
        Path="/acme"
```

and

```
Set-Cookie: Part_Number="Riding_Rocket_0023"; Version="1";
        Path="/acme/ammo"
```

A subsequent request by the user agent to the (same) server for URLs of the form /acme/ammo/... would include the following request header:

```
Cookie: $Version="1";
        Part_Number="Riding_Rocket_0023"; $Path="/acme/ammo";
        Part_Number="Rocket_Launcher_0001"; $Path="/acme"
```

Note that the NAME=VALUE pair for the cookie with the more specific Path attribute, /acme/ammo, comes before the one with the less specific Path attribute, /acme. Further note that the same cookie name appears more than once.

A subsequent request by the user agent to the (same) server for a URL of the form /acme/parts/ would include the following request header:

```
Cookie: $Version="1"; Part_Number="Rocket_Launcher_0001"; $Path="/acme"
```

Here, the second cookie's Path attribute /acme/ammo is not a prefix of the request URL, /acme/parts/, so the cookie does not get forwarded to the server.

6. IMPLEMENTATION CONSIDERATIONS

Here we speculate on likely or desirable details for an origin server that implements state management.

6.1 Set-Cookie Content

An origin server's content should probably be divided into disjoint application areas, some of which require the use of state information. The application areas can be distinguished by their request URLs. The Set-Cookie header can incorporate information about the application areas by setting the Path attribute for each one.

The session information can obviously be clear or encoded text that describes state. However, if it grows too large, it can become unwieldy. Therefore, an implementor might choose for the session information to be a key to a server-side resource. Of course, using

a database creates some problems that this state management specification was meant to avoid, namely:

1. keeping real state on the server side;

2. how and when to garbage-collect the database entry, in case the user agent terminates the session by, for example, exiting.

6.2 Stateless Pages

Caching benefits the scalability of WWW. Therefore it is important to reduce the number of documents that have state embedded in them inherently. For example, if a shopping-basket-style application always displays a user's current basket contents on each page, those pages cannot be cached, because each user's basket's contents would be different. On the other hand, if each page contains just a link that allows the user to "Look at My Shopping Basket", the page can be cached.

6.3 Implementation Limits

Practical user agent implementations have limits on the number and size of cookies that they can store. In general, user agents' cookie support should have no fixed limits. They should strive to store as many frequently-used cookies as possible. Furthermore, general-use user agents should provide each of the following minimum capabilities individually, although not necessarily simultaneously:

* at least 300 cookies

* at least 4096 bytes per cookie (as measured by the size of the characters that comprise the cookie non-terminal in the syntax description of the Set-Cookie header)

* at least 20 cookies per unique host or domain name

User agents created for specific purposes or for limited-capacity devices should provide at least 20 cookies of 4096 bytes, to ensure that the user can interact with a session-based origin server.

The information in a Set-Cookie response header must be retained in its entirety. If for some reason there is inadequate space to store the cookie, it must be discarded, not truncated.

Applications should use as few and as small cookies as possible, and they should cope gracefully with the loss of a cookie.

6.3.1 Denial of Service Attacks

User agents may choose to set an upper bound on the number of cookies to be stored from a given host or domain name or on the size of the cookie information. Otherwise a malicious server could attempt to flood a user agent with many cookies, or large cookies, on successive responses, which would force out cookies the user agent had received from other servers. However, the minima specified above should still be supported.

7. PRIVACY

7.1 User Agent Control

An origin server could create a Set-Cookie header to track the path of a user through the server. Users may object to this behavior as an intrusive accumulation of information, even if their identity is not evident. (Identity might become evident if a user subsequently fills out a form that contains identifying information.) This state management specification therefore requires that a user agent give the user control over such a possible intrusion, although the interface through which the user is given this control is left unspecified. However, the control mechanisms provided shall at least allow the user

* to completely disable the sending and saving of cookies.

* to determine whether a stateful session is in progress.

* to control the saving of a cookie on the basis of the cookie's Domain attribute.

Such control could be provided by, for example, mechanisms

* to notify the user when the user agent is about to send a cookie to the origin server, offering the option not to begin a session.

* to display a visual indication that a stateful session is in progress.

* to let the user decide which cookies, if any, should be saved when the user concludes a window or user agent session.

* to let the user examine the contents of a cookie at any time.

A user agent usually begins execution with no remembered state information. It should be possible to configure a user agent never to send Cookie headers, in which case it can never sustain state with

an origin server. (The user agent would then behave like one that is unaware of how to handle Set-Cookie response headers.)

When the user agent terminates execution, it should let the user discard all state information. Alternatively, the user agent may ask the user whether state information should be retained; the default should be "no". If the user chooses to retain state information, it would be restored the next time the user agent runs.

NOTE: User agents should probably be cautious about using files to store cookies long-term. If a user runs more than one instance of the user agent, the cookies could be commingled or otherwise messed up.

7.2 Protocol Design

The restrictions on the value of the Domain attribute, and the rules concerning unverifiable transactions, are meant to reduce the ways that cookies can "leak" to the "wrong" site. The intent is to restrict cookies to one, or a closely related set of hosts. Therefore a request-host is limited as to what values it can set for Domain. We consider it acceptable for hosts host1.foo.com and host2.foo.com to share cookies, but not a.com and b.com.

Similarly, a server can only set a Path for cookies that are related to the request-URI.

8. SECURITY CONSIDERATIONS

8.1 Clear Text

The information in the Set-Cookie and Cookie headers is unprotected. Two consequences are:

1. Any sensitive information that is conveyed in them is exposed to intruders.

2. A malicious intermediary could alter the headers as they travel in either direction, with unpredictable results.

These facts imply that information of a personal and/or financial nature should only be sent over a secure channel. For less sensitive information, or when the content of the header is a database key, an origin server should be vigilant to prevent a bad Cookie value from causing failures.

8.2 Cookie Spoofing

Proper application design can avoid spoofing attacks from related domains. Consider:

1. User agent makes request to victim.cracker.edu, gets back cookie session_id="1234" and sets the default domain victim.cracker.edu.

2. User agent makes request to spoof.cracker.edu, gets back cookie session-id="1111", with Domain=".cracker.edu".

3. User agent makes request to victim.cracker.edu again, and passes

   ```
   Cookie: $Version="1";
                   session_id="1234";
                   session_id="1111"; $Domain=".cracker.edu"
   ```

   The server at victim.cracker.edu should detect that the second cookie was not one it originated by noticing that the Domain attribute is not for itself and ignore it.

8.3 Unexpected Cookie Sharing

A user agent should make every attempt to prevent the sharing of session information between hosts that are in different domains. Embedded or inlined objects may cause particularly severe privacy problems if they can be used to share cookies between disparate hosts. For example, a malicious server could embed cookie information for host a.com in a URI for a CGI on host b.com. User agent implementors are strongly encouraged to prevent this sort of exchange whenever possible.

9. OTHER, SIMILAR, PROPOSALS

Three other proposals have been made to accomplish similar goals. This specification is an amalgam of Kristol's State-Info proposal and Netscape's Cookie proposal.

Brian Behlendorf proposed a Session-ID header that would be user-agent-initiated and could be used by an origin server to track "clicktrails". It would not carry any origin-server-defined state, however. Phillip Hallam-Baker has proposed another client-defined session ID mechanism for similar purposes.

While both session IDs and cookies can provide a way to sustain stateful sessions, their intended purpose is different, and, consequently, the privacy requirements for them are different. A user initiates session IDs to allow servers to track progress through them, or to distinguish multiple users on a shared machine. Cookies are server-initiated, so the cookie mechanism described here gives users control over something that would otherwise take place without the users' awareness. Furthermore, cookies convey rich, server-selected information, whereas session IDs comprise user-selected, simple information.

10. HISTORICAL

10.1 Compatibility With Netscape's Implementation

HTTP/1.0 clients and servers may use Set-Cookie and Cookie headers that reflect Netscape's original cookie proposal. These notes cover inter-operation between "old" and "new" cookies.

10.1.1 Extended Cookie Header

This proposal adds attribute-value pairs to the Cookie request header in a compatible way. An "old" client that receives a "new" cookie will ignore attributes it does not understand; it returns what it does understand to the origin server. A "new" client always sends cookies in the new form.

An "old" server that receives a "new" cookie will see what it thinks are many cookies with names that begin with a $, and it will ignore them. (The "old" server expects these cookies to be separated by semi-colon, not comma.) A "new" server can detect cookies that have passed through an "old" client, because they lack a $Version attribute.

10.1.2 Expires and Max-Age

Netscape's original proposal defined an Expires header that took a date value in a fixed-length variant format in place of Max-Age:

Wdy, DD-Mon-YY HH:MM:SS GMT

Note that the Expires date format contains embedded spaces, and that "old" cookies did not have quotes around values. Clients that implement to this specification should be aware of "old" cookies and Expires.

10.1.3 Punctuation

In Netscape's original proposal, the values in attribute-value pairs did not accept "-quoted strings. Origin servers should be cautious about sending values that require quotes unless they know the receiving user agent understands them (i.e., "new" cookies). A ("new") user agent should only use quotes around values in Cookie headers when the cookie's version(s) is (are) all compliant with this specification or later.

In Netscape's original proposal, no whitespace was permitted around the = that separates attribute-value pairs. Therefore such whitespace should be used with caution in new implementations.

10.2 Caching and HTTP/1.0

Some caches, such as those conforming to HTTP/1.0, will inevitably cache the Set-Cookie header, because there was no mechanism to suppress caching of headers prior to HTTP/1.1. This caching can lead to security problems. Documents transmitted by an origin server along with Set-Cookie headers will usually either be uncachable, or will be "pre-expired". As long as caches obey instructions not to cache documents (following Expires: <a date in the past> or Pragma: no-cache (HTTP/1.0), or Cache-control: no-cache (HTTP/1.1)) uncachable documents present no problem. However, pre-expired documents may be stored in caches. They require validation (a conditional GET) on each new request, but some cache operators loosen the rules for their caches, and sometimes serve expired documents without first validating them. This combination of factors can lead to cookies meant for one user later being sent to another user. The Set-Cookie header is stored in the cache, and, although the document is stale (expired), the cache returns the document in response to later requests, including cached headers.

11. ACKNOWLEDGEMENTS

This document really represents the collective efforts of the following people, in addition to the authors: Roy Fielding, Marc Hedlund, Ted Hardie, Koen Holtman, Shel Kaphan, Rohit Khare.

12. AUTHORS' ADDRESSES

David M. Kristol
Bell Laboratories, Lucent Technologies
600 Mountain Ave. Room 2A-227
Murray Hill, NJ 07974

Phone: (908) 582-2250
Fax: (908) 582-5809
EMail: dmk@bell-labs.com

Lou Montulli
Netscape Communications Corp.
501 E. Middlefield Rd.
Mountain View, CA 94043

Phone: (415) 528-2600
EMail: montulli@netscape.com

Network Working Group
Request for Comments: 1036
Obsoletes: RFC-850

M. Horton
AT&T Bell Laboratories
R. Adams
Center for Seismic Studies
December 1987

Standard for Interchange of USENET Messages

STATUS OF THIS MEMO

This document defines the standard format for the interchange of network News messages among USENET hosts. It updates and replaces RFC-850, reflecting version B2.11 of the News program. This memo is disributed as an RFC to make this information easily accessible to the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

1. Introduction

This document defines the standard format for the interchange of network News messages among USENET hosts. It describes the format for messages themselves and gives partial standards for transmission of news. The news transmission is not entirely in order to give a good deal of flexibility to the hosts to choose transmission hardware and software, to batch news, and so on.

There are five sections to this document. Section two defines the format. Section three defines the valid control messages. Section four specifies some valid transmission methods. Section five describes the overall news propagation algorithm.

2. Message Format

The primary consideration in choosing a message format is that it fit in with existing tools as well as possible. Existing tools include implementations of both mail and news. (The notesfiles system from the University of Illinois is considered a news implementation.) A standard format for mail messages has existed for many years on the Internet, and this format meets most of the needs of USENET. Since the Internet format is extensible, extensions to meet the additional needs of USENET are easily made within the Internet standard. Therefore, the rule is adopted that all USENET news messages must be formatted as valid Internet mail messages, according to the Internet standard RFC-822. The USENET News standard is more restrictive than the Internet standard,

placing additional requirements on each message and forbidding use of certain Internet features. However, it should always be possible to use a tool expecting an Internet message to process a news message. In any situation where this standard conflicts with the Internet standard, RFC-822 should be considered correct and this standard in error.

Here is an example USENET message to illustrate the fields.

```
From: jerry@eagle.ATT.COM (Jerry Schwarz)
Path: cbosgd!mhuxj!mhuxt!eagle!jerry
Newsgroups: news.announce
Subject: Usenet Etiquette -- Please Read
Message-ID: <642@eagle.ATT.COM>
Date: Fri, 19 Nov 82 16:14:55 GMT
Followup-To: news.misc
Expires: Sat, 1 Jan 83 00:00:00 -0500
Organization: AT&T Bell Laboratories, Murray Hill

The body of the message comes here, after a blank line.
```

Here is an example of a message in the old format (before the existence of this standard). It is recommended that implementations also accept messages in this format to ease upward conversion.

```
From: cbosgd!mhuxj!mhuxt!eagle!jerry (Jerry Schwarz)
Newsgroups: news.misc
Title: Usenet Etiquette -- Please Read
Article-I.D.: eagle.642
Posted: Fri Nov 19 16:14:55 1982
Received: Fri Nov 19 16:59:30 1982
Expires: Mon Jan 1 00:00:00 1990

The body of the message comes here, after a blank line.
```

Some news systems transmit news in the A format, which looks like this:

```
Aeagle.642
news.misc
cbosgd!mhuxj!mhuxt!eagle!jerry
Fri Nov 19 16:14:55 1982
Usenet Etiquette - Please Read
The body of the message comes here, with no blank line.
```

A standard USENET message consists of several header lines, followed by a blank line, followed by the body of the message. Each header

line consist of a keyword, a colon, a blank, and some additional information. This is a subset of the Internet standard, simplified to allow simpler software to handle it. The "From" line may optionally include a full name, in the format above, or use the Internet angle bracket syntax. To keep the implementations simple, other formats (for example, with part of the machine address after the close parenthesis) are not allowed. The Internet convention of continuation header lines (beginning with a blank or tab) is allowed.

Certain headers are required, and certain other headers are optional. Any unrecognized headers are allowed, and will be passed through unchanged. The required header lines are "From", "Date", "Newsgroups", "Subject", "Message-ID", and "Path". The optional header lines are "Followup-To", "Expires", "Reply-To", "Sender", "References", "Control", "Distribution", "Keywords", "Summary", "Approved", "Lines", "Xref", and "Organization". Each of these header lines will be described below.

2.1. Required Header lines

2.1.1. From

The "From" line contains the electronic mailing address of the person who sent the message, in the Internet syntax. It may optionally also contain the full name of the person, in parentheses, after the electronic address. The electronic address is the same as the entity responsible for originating the message, unless the "Sender" header is present, in which case the "From" header might not be verified. Note that in all host and domain names, upper and lower case are considered the same, thus "mark@cbosgd.ATT.COM", "mark@cbosgd.att.com", and "mark@CBosgD.ATt.COm" are all equivalent. User names may or may not be case sensitive, for example, "Billy@cbosgd.ATT.COM" might be different from "BillY@cbosgd.ATT.COM". Programs should avoid changing the case of electronic addresses when forwarding news or mail.

RFC-822 specifies that all text in parentheses is to be interpreted as a comment. It is common in Internet mail to place the full name of the user in a comment at the end of the "From" line. This standard specifies a more rigid syntax. The full name is not considered a comment, but an optional part of the header line. Either the full name is omitted, or it appears in parentheses after the electronic address of the person posting the message, or it appears before an electronic address which is enclosed in angle brackets. Thus, the three permissible forms are:


```
From: mark@cbosgd.ATT.COM
From: mark@cbosgd.ATT.COM (Mark Horton)
From: Mark Horton <mark@cbosgd.ATT.COM>
```

Full names may contain any printing ASCII characters from space through tilde, except that they may not contain "(" (left parenthesis), ")" (right parenthesis), "<" (left angle bracket), or ">" (right angle bracket). Additional restrictions may be placed on full names by the mail standard, in particular, the characters "," (comma), ":" (colon), "@" (at), "!" (bang), "/" (slash), "=" (equal), and ";" (semicolon) are inadvisable in full names.

2.1.2. Date

The "Date" line (formerly "Posted") is the date that the message was originally posted to the network. Its format must be acceptable both in RFC-822 and to the getdate(3) routine that is provided with the Usenet software. This date remains unchanged as the message is propagated throughout the network. One format that is acceptable to both is:

```
Wdy, DD Mon YY HH:MM:SS TIMEZONE
```

Several examples of valid dates appear in the sample message above. Note in particular that ctime(3) format:

```
Wdy Mon DD HH:MM:SS YYYY
```

is not acceptable because it is not a valid RFC-822 date. However, since older software still generates this format, news implementations are encouraged to accept this format and translate it into an acceptable format.

There is no hope of having a complete list of timezones. Universal Time (GMT), the North American timezones (PST, PDT, MST, MDT, CST, CDT, EST, EDT) and the +/-hhmm offset specifed in RFC-822 should be supported. It is recommended that times in message headers be transmitted in GMT and displayed in the local time zone.

2.1.3. Newsgroups

The "Newsgroups" line specifies the newsgroup or newsgroups in which the message belongs. Multiple newsgroups may be specified, separated by a comma. Newsgroups specified must all be the names of existing newsgroups, as no new newsgroups will be created by simply posting to them.

Wildcards (e.g., the word "all") are never allowed in a "Newsgroups" line. For example, a newsgroup comp.all is illegal, although a newsgroup rec.sport.football is permitted.

If a message is received with a "Newsgroups" line listing some valid newsgroups and some invalid newsgroups, a host should not remove invalid newsgroups from the list. Instead, the invalid newsgroups should be ignored. For example, suppose host A subscribes to the classes btl.all and comp.all, and exchanges news messages with host B, which subscribes to comp.all but not btl.all. Suppose A receives a message with Newsgroups: comp.unix,btl.general.

This message is passed on to B because B receives comp.unix, but B does not receive btl.general. A must leave the "Newsgroups" line unchanged. If it were to remove btl.general, the edited header could eventually re-enter the btl.all class, resulting in a message that is not shown to users subscribing to btl.general. Also, follow-ups from outside btl.all would not be shown to such users.

2.1.4. Subject

The "Subject" line (formerly "Title") tells what the message is about. It should be suggestive enough of the contents of the message to enable a reader to make a decision whether to read the message based on the subject alone. If the message is submitted in response to another message (e.g., is a follow-up) the default subject should begin with the four characters "Re:", and the "References" line is required. For follow-ups, the use of the "Summary" line is encouraged.

2.1.5. Message-ID

The "Message-ID" line gives the message a unique identifier. The Message-ID may not be reused during the lifetime of any previous message with the same Message-ID. (It is recommended that no Message-ID be reused for at least two years.) Message-ID's have the syntax:

<string not containing blank or ">">

In order to conform to RFC-822, the Message-ID must have the format:

<unique@full_domain_name>

where full_domain_name is the full name of the host at which the message entered the network, including a domain that host is in, and unique is any string of printing ASCII characters, not including "<" (left angle bracket), ">" (right angle bracket), or "@" (at sign).

For example, the unique part could be an integer representing a sequence number for messages submitted to the network, or a short string derived from the date and time the message was created. For example, a valid Message-ID for a message submitted from host ucbvax in domain "Berkeley.EDU" would be "<4123@ucbvax.Berkeley.EDU>". Programmers are urged not to make assumptions about the content of Message-ID fields from other hosts, but to treat them as unknown character strings. It is not safe, for example, to assume that a Message-ID will be under 14 characters, that it is unique in the first 14 characters, nor that is does not contain a "/".

The angle brackets are considered part of the Message-ID. Thus, in references to the Message-ID, such as the ihave/sendme and cancel control messages, the angle brackets are included. White space characters (e.g., blank and tab) are not allowed in a Message-ID. Slashes ("/") are strongly discouraged. All characters between the angle brackets must be printing ASCII characters.

2.1.6. Path

This line shows the path the message took to reach the current system. When a system forwards the message, it should add its own name to the list of systems in the "Path" line. The names may be separated by any punctuation character or characters (except "." which is considered part of the hostname). Thus, the following are valid entries:

```
cbosgd!mhuxj!mhuxt
cbosgd, mhuxj, mhuxt
@cbosgd.ATT.COM,@mhuxj.ATT.COM,@mhuxt.ATT.COM
teklabs, zehntel, sri-unix@cca!decvax
```

(The latter path indicates a message that passed through decvax, cca, sri-unix, zehntel, and teklabs, in that order.) Additional names should be added from the left. For example, the most recently added name in the fourth example was teklabs. Letters, digits, periods and hyphens are considered part of host names; other punctuation, including blanks, are considered separators.

Normally, the rightmost name will be the name of the originating system. However, it is also permissible to include an extra entry on the right, which is the name of the sender. This is for upward compatibility with older systems.

The "Path" line is not used for replies, and should not be taken as a mailing address. It is intended to show the route the message traveled to reach the local host. There are several uses for this information. One is to monitor USENET routing for performance

reasons. Another is to establish a path to reach new hosts. Perhaps the most important use is to cut down on redundant USENET traffic by failing to forward a message to a host that is known to have already received it. In particular, when host A sends a message to host B, the "Path" line includes A, so that host B will not immediately send the message back to host A. The name each host uses to identify itself should be the same as the name by which its neighbors know it, in order to make this optimization possible.

A host adds its own name to the front of a path when it receives a message from another host. Thus, if a message with path "A!X!Y!Z" is passed from host A to host B, B will add its own name to the path when it receives the message from A, e.g., "B!A!X!Y!Z". If B then passes the message on to C, the message sent to C will contain the path "B!A!X!Y!Z", and when C receives it, C will change it to "C!B!A!X!Y!Z".

Special upward compatibility note: Since the "From", "Sender", and "Reply-To" lines are in Internet format, and since many USENET hosts do not yet have mailers capable of understanding Internet format, it would break the reply capability to completely sever the connection between the "Path" header and the reply function. It is recognized that the path is not always a valid reply string in older implementations, and no requirement to fix this problem is placed on implementations. However, the existing convention of placing the host name and an "!" at the front of the path, and of starting the path with the host name, an "!", and the user name, should be maintained when possible.

2.2. Optional Headers

2.2.1. Reply-To

This line has the same format as "From". If present, mailed replies to the author should be sent to the name given here. Otherwise, replies are mailed to the name on the "From" line. (This does not prevent additional copies from being sent to recipients named by the replier, or on "To" or "Cc" lines.) The full name may be optionally given, in parentheses, as in the "From" line.

2.2.2. Sender

This field is present only if the submitter manually enters a "From" line. It is intended to record the entity responsible for submitting the message to the network. It should be verified by the software at the submitting host.

For example, if John Smith is visiting CCA and wishes to post a message to the network, using friend Sarah Jones' account, the message might read:

```
From: smith@ucbvax.Berkeley.EDU (John Smith)
Sender: jones@cca.COM (Sarah Jones)
```

If a gateway program enters a mail message into the network at host unix.SRI.COM, the lines might read:

```
From: John.Doe@A.CS.CMU.EDU
Sender: network@unix.SRI.COM
```

The primary purpose of this field is to be able to track down messages to determine how they were entered into the network. The full name may be optionally given, in parentheses, as in the "From" line.

2.2.3. Followup-To

This line has the same format as "Newsgroups". If present, follow-up messages are to be posted to the newsgroup or newsgroups listed here. If this line is not present, follow-ups are posted to the newsgroup or newsgroups listed in the "Newsgroups" line.

If the keyword poster is present, follow-up messages are not permitted. The message should be mailed to the submitter of the message via mail.

2.2.4. Expires

This line, if present, is in a legal USENET date format. It specifies a suggested expiration date for the message. If not present, the local default expiration date is used. This field is intended to be used to clean up messages with a limited usefulness, or to keep important messages around for longer than usual. For example, a message announcing an upcoming seminar could have an expiration date the day after the seminar, since the message is not useful after the seminar is over. Since local hosts have local policies for expiration of news (depending on available disk space, for instance), users are discouraged from providing expiration dates for messages unless there is a natural expiration date associated with the topic. System software should almost never provide a default "Expires" line. Leave it out and allow local policies to be used unless there is a good reason not to.

2.2.5. References

This field lists the Message-ID's of any messages prompting the submission of this message. It is required for all follow-up messages, and forbidden when a new subject is raised. Implementations should provide a follow-up command, which allows a user to post a follow-up message. This command should generate a "Subject" line which is the same as the original message, except that if the original subject does not begin with "Re:" or "re:", the four characters "Re:" are inserted before the subject. If there is no "References" line on the original header, the "References" line should contain the Message-ID of the original message (including the angle brackets). If the original message does have a "References" line, the follow-up message should have a "References" line containing the text of the original "References" line, a blank, and the Message-ID of the original message.

The purpose of the "References" header is to allow messages to be grouped into conversations by the user interface program. This allows conversations within a newsgroup to be kept together, and potentially users might shut off entire conversations without unsubscribing to a newsgroup. User interfaces need not make use of this header, but all automatically generated follow-ups should generate the "References" line for the benefit of systems that do use it, and manually generated follow-ups (e.g., typed in well after the original message has been printed by the machine) should be encouraged to include them as well.

It is permissible to not include the entire previous "References" line if it is too long. An attempt should be made to include a reasonable number of backwards references.

2.2.6. Control

If a message contains a "Control" line, the message is a control message. Control messages are used for communication among USENET host machines, not to be read by users. Control messages are distributed by the same newsgroup mechanism as ordinary messages. The body of the "Control" header line is the message to the host.

For upward compatibility, messages that match the newsgroup pattern "all.all.ctl" should also be interpreted as control messages. If no "Control" header is present on such messages, the subject is used as the control message. However, messages on newsgroups matching this pattern do not conform to this standard.

Also for upward compatibility, if the first 4 characters of the "Subject:" line are "cmsg", the rest of the "Subject:" line should be interpreted as a control message.

2.2.7. Distribution

This line is used to alter the distribution scope of the message. It is a comma separated list similar to the "Newsgroups" line. User subscriptions are still controlled by "Newsgroups", but the message is sent to all systems subscribing to the newsgroups on the "Distribution" line in addition to the "Newsgroups" line. For the message to be transmitted, the receiving site must normally receive one of the specified newsgroups AND must receive one of the specified distributions. Thus, a message concerning a car for sale in New Jersey might have headers including:

```
Newsgroups: rec.auto,misc.forsale
Distribution: nj,ny
```

so that it would only go to persons subscribing to rec.auto or misc. for sale within New Jersey or New York. The intent of this header is to restrict the distribution of a newsgroup further, not to increase it. A local newsgroup, such as nj.crazy-eddie, will probably not be propagated by hosts outside New Jersey that do not show such a newsgroup as valid. A follow-up message should default to the same "Distribution" line as the original message, but the user can change it to a more limited one, or escalate the distribution if it was originally restricted and a more widely distributed reply is appropriate.

2.2.8. Organization

The text of this line is a short phrase describing the organization to which the sender belongs, or to which the machine belongs. The intent of this line is to help identify the person posting the message, since host names are often cryptic enough to make it hard to recognize the organization by the electronic address.

2.2.9. Keywords

A few well-selected keywords identifying the message should be on this line. This is used as an aid in determining if this message is interesting to the reader.

2.2.10. Summary

This line should contain a brief summary of the message. It is usually used as part of a follow-up to another message. Again, it

is very useful to the reader in determining whether to read the message.

2.2.11. Approved

This line is required for any message posted to a moderated newsgroup. It should be added by the moderator and consist of his mail address. It is also required with certain control messages.

2.2.12. Lines

This contains a count of the number of lines in the body of the message.

2.2.13. Xref

This line contains the name of the host (with domains omitted) and a white space separated list of colon-separated pairs of newsgroup names and message numbers. These are the newsgroups listed in the "Newsgroups" line and the corresponding message numbers from the spool directory.

This is only of value to the local system, so it should not be transmitted. For example, in:

```
Path: seismo!lll-crg!lll-lcc!pyramid!decwrl!reid
From: reid@decwrl.DEC.COM (Brian Reid)
Newsgroups: news.lists,news.groups
Subject: USENET READERSHIP SUMMARY REPORT FOR SEP 86
Message-ID: <5658@decwrl.DEC.COM>
Date: 1 Oct 86 11:26:15 GMT
Organization: DEC Western Research Laboratory
Lines: 441
Approved: reid@decwrl.UUCP
Xref: seismo news.lists:461 news.groups:6378
```

the "Xref" line shows that the message is message number 461 in the newsgroup news.lists, and message number 6378 in the newsgroup news.groups, on host seismo. This information may be used by certain user interfaces.

3. Control Messages

This section lists the control messages currently defined. The body of the "Control" header line is the control message. Messages are a sequence of zero or more words, separated by white space (blanks or tabs). The first word is the name of the control message, remaining words are parameters to the message. The remainder of the header

and the body of the message are also potential parameters; for example, the "From" line might suggest an address to which a response is to be mailed.

Implementors and administrators may choose to allow control messages to be carried out automatically, or to queue them for annual processing. However, manually processed messages should be dealt with promptly.

Failed control messages should NOT be mailed to the originator of the message, but to the local "usenet" account.

3.1. Cancel

```
cancel <Message-ID>
```

If a message with the given Message-ID is present on the local system, the message is cancelled. This mechanism allows a user to cancel a message after the message has been distributed over the network.

If the system is unable to cancel the message as requested, it should not forward the cancellation request to its neighbor systems.

Only the author of the message or the local news administrator is allowed to send this message. The verified sender of a message is the "Sender" line, or if no "Sender" line is present, the "From" line. The verified sender of the cancel message must be the same as either the "Sender" or "From" field of the original message. A verified sender in the cancel message is allowed to match an unverified "From" in the original message.

3.2. Ihave/Sendme

```
ihave <Message-ID list> [<remotesys>]
sendme <Message-ID list> [<remotesys>]
```

This message is part of the ihave/sendme protocol, which allows one host (say A) to tell another host (B) that a particular message has been received on A. Suppose that host A receives message "<1234@ucbvax.Berkeley.edu>", and wishes to transmit the message to host B.

A sends the control message "ihave <1234@ucbvax.Berkeley.edu> A" to host B (by posting it to newsgroup to.B). B responds with the control message "sendme <1234@ucbvax.Berkeley.edu> B" (on newsgroup to.A), if it has not already received the message. Upon receiving

the sendme message, A sends the message to B.

This protocol can be used to cut down on redundant traffic between hosts. It is optional and should be used only if the particular situation makes it worthwhile. Frequently, the outcome is that, since most original messages are short, and since there is a high overhead to start sending a new message with UUCP, it costs as much to send the ihave as it would cost to send the message itself.

One possible solution to this overhead problem is to batch requests. Several Message-ID's may be announced or requested in one message. If no Message-ID's are listed in the control message, the body of the message should be scanned for Message-ID's, one per line.

3.3. Newgroup

newgroup <groupname> [moderated]

This control message creates a new newsgroup with the given name. Since no messages may be posted or forwarded until a newsgroup is created, this message is required before a newsgroup can be used. The body of the message is expected to be a short paragraph describing the intended use of the newsgroup.

If the second argument is present and it is the keyword moderated, the group should be created moderated instead of the default of unmoderated. The newgroup message should be ignored unless there is an "Approved" line in the same message header.

3.4. Rmgroup

rmgroup <groupname>

This message removes a newsgroup with the given name. Since the newsgroup is removed from every host on the network, this command should be used carefully by a responsible administrator. The rmgroup message should be ignored unless there is an "Approved:" line in the same message header.

3.5. Sendsys

sendsys (no arguments)

The sys file, listing all neighbors and the newsgroups to be sent to each neighbor, will be mailed to the author of the control message ("Reply-To", if present, otherwise "From"). This information is considered public information, and it is a requirement of membership in USENET that this information be provided on request, either automatically in response to this control message, or manually, by mailing the requested information to the author of the message. This information is used to keep the map of USENET up to date, and to determine where netnews is sent.

The format of the file mailed back to the author should be the same as that of the sys file. This format has one line per neighboring host (plus one line for the local host), containing four colon separated fields. The first field has the host name of the neighbor, the second field has a newsgroup pattern describing the newsgroups sent to the neighbor. The third and fourth fields are not defined by this standard. The sys file is not the same as the UUCP L.sys file. A sample response is:

```
From: cbosgd!mark  (Mark Horton)
Date: Sun, 27 Mar 83 20:39:37 -0500
Subject: response to your sendsys request
To: mark@cbosgd.ATT.COM

Responding-System: cbosgd.ATT.COM
cbosgd:osg,cb,btl,bell,world,comp,sci,rec,talk,misc,news,soc,to,
      test
ucbvax:world,comp,to.ucbvax:L:
cbosg:world,comp,bell,btl,cb,osg,to.cbosg:F:/usr/spool/outnews
      /cbosg
cbosgb:osg,to.cbosgb:F:/usr/spool/outnews/cbosgb
sescent:world,comp,bell,btl,cb,to.sescent:F:/usr/spool/outnews
      /sescent
npois:world,comp,bell,btl,ug,to.npois:F:/usr/spool/outnews/npois
mhuxi:world,comp,bell,btl,ug,to.mhuxi:F:/usr/spool/outnews/mhuxi
```

3.6. Version

version (no arguments)

The name and version of the software running on the local system is to be mailed back to the author of the message ("Reply-to" if present, otherwise "From").

3.7. Checkgroups

The message body is a list of "official" newsgroups and their description, one group per line. They are compared against the list of active newsgroups on the current host. The names of any obsolete or new newsgroups are mailed to the user "usenet" and descriptions of the new newsgroups are added to the help file used when posting news.

4. Transmission Methods

USENET is not a physical network, but rather a logical network resting on top of several existing physical networks. These networks include, but are not limited to, UUCP, the Internet, an Ethernet, the BLICN network, an NSC Hyperchannel, and a BERKNET. What is important is that two neighboring systems on USENET have some method to get a new message, in the format listed here, from one system to the other, and once on the receiving system, processed by the netnews software on that system. (On UNIX systems, this usually means the rnews program being run with the message on the standard input. <1>)

It is not a requirement that USENET hosts have mail systems capable of understanding the Internet mail syntax, but it is strongly recommended. Since "From", "Reply-To", and "Sender" lines use the Internet syntax, replies will be difficult or impossible without an Internet mailer. A host without an Internet mailer can attempt to use the "Path" header line for replies, but this field is not guaranteed to be a working path for replies. In any event, any host generating or forwarding news messages must have an Internet address that allows them to receive mail from hosts with Internet mailers, and they must include their Internet address on their From line.

4.1. Remote Execution

Some networks permit direct remote command execution. On these networks, news may be forwarded by spooling the rnews command with the message on the standard input. For example, if the remote system is called remote, news would be sent over a UUCP link with the command:

```
uux - remote!rnews
```

and on a Berknet:

```
net -mremote rnews
```


It is important that the message be sent via a reliable mechanism, normally involving the possibility of spooling, rather than direct real-time remote execution. This is because, if the remote system is down, a direct execution command will fail, and the message will never be delivered. If the message is spooled, it will eventually be delivered when both systems are up.

4.2. Transfer by Mail

On some systems, direct remote spooled execution is not possible. However, most systems support electronic mail, and a news message can be sent as mail. One approach is to send a mail message which is identical to the news message: the mail headers are the news headers, and the mail body is the news body. By convention, this mail is sent to the user newsmail on the remote machine.

One problem with this method is that it may not be possible to convince the mail system that the "From" line of the message is valid, since the mail message was generated by a program on a system different from the source of the news message. Another problem is that error messages caused by the mail transmission would be sent to the originator of the news message, who has no control over news transmission between two cooperating hosts and does not know whom to contact. Transmission error messages should be directed to a responsible contact person on the sending machine.

A solution to this problem is to encapsulate the news message into a mail message, such that the entire message (headers and body) are part of the body of the mail message. The convention here is that such mail is sent to user rnews on the remote system. A mail message body is generated by prepending the letter N to each line of the news message, and then attaching whatever mail headers are convenient to generate. The N's are attached to prevent any special lines in the news message from interfering with mail transmission, and to prevent any extra lines inserted by the mailer (headers, blank lines, etc.) from becoming part of the news message. A program on the receiving machine receives mail to rnews, extracting the message itself and invoking the rnews program. An example in this format might look like this:

```
Date: Mon, 3 Jan 83 08:33:47 MST
From: news@cbosgd.ATT.COM
Subject: network news message
To: rnews@npois.ATT.COM

NPath: cbosgd!mhuxj!harpo!utah-cs!sask!derek
NFrom: derek@sask.UUCP (Derek Andrew)
NNewsgroups: misc.test
NSubject: necessary test
NMessage-ID: <176@sask.UUCP>
NDate: Mon, 3 Jan 83 00:59:15 MST
N
NThis really is a test.  If anyone out there more than 6
Nhops away would kindly confirm this note I would
Nappreciate it.  We suspect that our news postings
Nare not getting out into the world.
N
```

Using mail solves the spooling problem, since mail must always be spooled if the destination host is down. However, it adds more overhead to the transmission process (to encapsulate and extract the message) and makes it harder for software to give different priorities to news and mail.

4.3. Batching

Since news messages are usually short, and since a large number of messages are often sent between two hosts in a day, it may make sense to batch news messages. Several messages can be combined into one large message, using conventions agreed upon in advance by the two hosts. One such batching scheme is described here; its use is highly recommended.

News messages are combined into a script, separated by a header of the form:

```
#! rnews 1234
```

where 1234 is the length of the message in bytes. Each such line is followed by a message containing the given number of bytes. (The newline at the end of each line of the message is counted as one byte, for purposes of this count, even if it is stored as <CARRIAGE RETURN><LINE FEED>.) For example, a batch of message might look like this:


```
#! rnews 239
From: jerry@eagle.ATT.COM (Jerry Schwarz)
Path: cbosgd!mhuxj!mhuxt!eagle!jerry
Newsgroups: news.announce
Subject: Usenet Etiquette -- Please Read
Message-ID: <642@eagle.ATT.COM>
Date: Fri, 19 Nov 82 16:14:55 EST
Approved: mark@cbosgd.ATT.COM

Here is an important message about USENET Etiquette.
#! rnews 234
From: jerry@eagle.ATT.COM (Jerry Schwarz)
Path: cbosgd!mhuxj!mhuxt!eagle!jerry
Newsgroups: news.announce
Subject: Notes on Etiquette message
Message-ID: <643@eagle.ATT.COM>
Date: Fri, 19 Nov 82 17:24:12 EST
Approved: mark@cbosgd.ATT.COM

There was something I forgot to mention in the last
message.
```

Batched news is recognized because the first character in the message is #. The message is then passed to the unbatcher for interpretation.

The second argument (in this example rnews) determines which batching scheme is being used. Cooperating hosts may use whatever scheme is appropriate for them.

5. The News Propagation Algorithm

This section describes the overall scheme of USENET and the algorithm followed by hosts in propagating news to the entire logical network. Since all hosts are affected by incorrectly formatted messages and by propagation errors, it is important for the method to be standardized.

USENET is a directed graph. Each node in the graph is a host computer, and each arc in the graph is a transmission path from one host to another host. Each arc is labeled with a newsgroup pattern, specifying which newsgroup classes are forwarded along that link. Most arcs are bidirectional, that is, if host A sends a class of newsgroups to host B, then host B usually sends the same class of newsgroups to host A. This bidirectionality is not, however, required.

USENET is made up of many subnetworks. Each subnet has a name, such

as comp or btl. Each subnet is a connected graph, that is, a path exists from every node to every other node in the subnet. In addition, the entire graph is (theoretically) connected. (In practice, some political considerations have caused some hosts to be unable to post messages reaching the rest of the network.)

A message is posted on one machine to a list of newsgroups. That machine accepts it locally, then forwards it to all its neighbors that are interested in at least one of the newsgroups of the message. (Site A deems host B to be "interested" in a newsgroup if the newsgroup matches the pattern on the arc from A to B. This pattern is stored in a file on the A machine.) The hosts receiving the incoming message examine it to make sure they really want the message, accept it locally, and then in turn forward the message to all their interested neighbors. This process continues until the entire network has seen the message.

An important part of the algorithm is the prevention of loops. The above process would cause a message to loop along a cycle forever. In particular, when host A sends a message to host B, host B will send it back to host A, which will send it to host B, and so on. One solution to this is the history mechanism. Each host keeps track of all messages it has seen (by their Message-ID) and whenever a message comes in that it has already seen, the incoming message is discarded immediately. This solution is sufficient to prevent loops, but additional optimizations can be made to avoid sending messages to hosts that will simply throw them away.

One optimization is that a message should never be sent to a machine listed in the "Path" line of the header. When a machine name is in the "Path" line, the message is known to have passed through the machine. Another optimization is that, if the message originated on host A, then host A has already seen the message. Thus, if a message is posted to newsgroup misc.misc, it will match the pattern misc.all (where all is a metasymbol that matches any string), and will be forwarded to all hosts that subscribe to misc.all (as determined by what their neighbors send them). These hosts make up the misc subnetwork. A message posted to btl.general will reach all hosts receiving btl.all, but will not reach hosts that do not get btl.all. In effect, the messages reaches the btl subnetwork. A messages posted to newsgroups misc.misc,btl.general will reach all hosts subscribing to either of the two classes.

Notes

<1> UNIX is a registered trademark of AT&T.

Network Working Group
Request for Comments: 2068
Category: Standards Track

R. Fielding
UC Irvine
J. Gettys
J. Mogul
DEC
H. Frystyk
T. Berners-Lee
MIT/LCS
January 1997

Hypertext Transfer Protocol -- HTTP/1.1

Status of this Memo

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

HTTP has been in use by the World-Wide Web global information initiative since 1990. This specification defines the protocol referred to as "HTTP/1.1".

Table of Contents

1 Introduction....7
1.1 Purpose....7
1.2 Requirements....7
1.3 Terminology....8
1.4 Overall Operation....11
2 Notational Conventions and Generic Grammar....13
2.1 Augmented BNF....13
2.2 Basic Rules....15
3 Protocol Parameters....17
3.1 HTTP Version....17

3.2 Uniform Resource Identifiers....18
3.2.1 General Syntax....18
3.2.2 http URL....19
3.2.3 URI Comparison....20
3.3 Date/Time Formats....21
3.3.1 Full Date....21
3.3.2 Delta Seconds....22
3.4 Character Sets....22
3.5 Content Codings....23
3.6 Transfer Codings....24
3.7 Media Types....25
3.7.1 Canonicalization and Text Defaults....26
3.7.2 Multipart Types....27
3.8 Product Tokens....28
3.9 Quality Values....28
3.10 Language Tags....28
3.11 Entity Tags....29
3.12 Range Units....30
4 HTTP Message....30
4.1 Message Types....30
4.2 Message Headers....31
4.3 Message Body....32
4.4 Message Length....32
4.5 General Header Fields....34
5 Request....34
5.1 Request-Line....34
5.1.1 Method....35
5.1.2 Request-URI....35
5.2 The Resource Identified by a Request....37
5.3 Request Header Fields....37
6 Response....38
6.1 Status-Line....38
6.1.1 Status Code and Reason Phrase....39
6.2 Response Header Fields....41
7 Entity....41
7.1 Entity Header Fields....41
7.2 Entity Body....42
7.2.1 Type....42
7.2.2 Length....43
8 Connections....43
8.1 Persistent Connections....43
8.1.1 Purpose....43
8.1.2 Overall Operation....44
8.1.3 Proxy Servers....45
8.1.4 Practical Considerations....45
8.2 Message Transmission Requirements....46
9 Method Definitions....48
9.1 Safe and Idempotent Methods....48

9.1.1 Safe Methods 48
9.1.2 Idempotent Methods 49
9.2 OPTIONS 49
9.3 GET 50
9.4 HEAD 50
9.5 POST 51
9.6 PUT 52
9.7 DELETE 53
9.8 TRACE 53
10 Status Code Definitions....53
10.1 Informational 1xx 54
10.1.1 100 Continue 54
10.1.2 101 Switching Protocols 54
10.2 Successful 2xx 54
10.2.1 200 OK 54
10.2.2 201 Created 55
10.2.3 202 Accepted 55
10.2.4 203 Non-Authoritative Information 55
10.2.5 204 No Content 55
10.2.6 205 Reset Content 56
10.2.7 206 Partial Content 56
10.3 Redirection 3xx 56
10.3.1 300 Multiple Choices 57
10.3.2 301 Moved Permanently 57
10.3.3 302 Moved Temporarily 58
10.3.4 303 See Other 58
10.3.5 304 Not Modified 58
10.3.6 305 Use Proxy 59
10.4 Client Error 4xx 59
10.4.1 400 Bad Request 60
10.4.2 401 Unauthorized 60
10.4.3 402 Payment Required 60
10.4.4 403 Forbidden 60
10.4.5 404 Not Found 60
10.4.6 405 Method Not Allowed 61
10.4.7 406 Not Acceptable 61
10.4.8 407 Proxy Authentication Required 61
10.4.9 408 Request Timeout 62
10.4.10 409 Conflict 62
10.4.11 410 Gone 62
10.4.12 411 Length Required 63
10.4.13 412 Precondition Failed 63
10.4.14 413 Request Entity Too Large 63
10.4.15 414 Request-URI Too Long 63
10.4.16 415 Unsupported Media Type 63
10.5 Server Error 5xx 64
10.5.1 500 Internal Server Error 64
10.5.2 501 Not Implemented 64

10.5.3 502 Bad Gateway 64
10.5.4 503 Service Unavailable 64
10.5.5 504 Gateway Timeout 64
10.5.6 505 HTTP Version Not Supported 65
11 Access Authentication....65
11.1 Basic Authentication Scheme 66
11.2 Digest Authentication Scheme 67
12 Content Negotiation....67
12.1 Server-driven Negotiation 68
12.2 Agent-driven Negotiation 69
12.3 Transparent Negotiation 70
13 Caching in HTTP....70
13.1.1 Cache Correctness 72
13.1.2 Warnings 73
13.1.3 Cache-control Mechanisms 74
13.1.4 Explicit User Agent Warnings 74
13.1.5 Exceptions to the Rules and Warnings 75
13.1.6 Client-controlled Behavior 75
13.2 Expiration Model 75
13.2.1 Server-Specified Expiration 75
13.2.2 Heuristic Expiration 76
13.2.3 Age Calculations 77
13.2.4 Expiration Calculations 79
13.2.5 Disambiguating Expiration Values 80
13.2.6 Disambiguating Multiple Responses 80
13.3 Validation Model 81
13.3.1 Last-modified Dates 82
13.3.2 Entity Tag Cache Validators 82
13.3.3 Weak and Strong Validators 82
13.3.4 Rules for When to Use Entity Tags and Last-modified Dates....85
13.3.5 Non-validating Conditionals 86
13.4 Response Cachability 86
13.5 Constructing Responses From Caches 87
13.5.1 End-to-end and Hop-by-hop Headers 88
13.5.2 Non-modifiable Headers 88
13.5.3 Combining Headers 89
13.5.4 Combining Byte Ranges 90
13.6 Caching Negotiated Responses 90
13.7 Shared and Non-Shared Caches 91
13.8 Errors or Incomplete Response Cache Behavior 91
13.9 Side Effects of GET and HEAD 92
13.10 Invalidation After Updates or Deletions 92
13.11 Write-Through Mandatory 93
13.12 Cache Replacement 93
13.13 History Lists 93
14 Header Field Definitions....94
14.1 Accept 95

14.2 Accept-Charset 97
14.3 Accept-Encoding 97
14.4 Accept-Language 98
14.5 Accept-Ranges 99
14.6 Age 99
14.7 Allow 100
14.8 Authorization 100
14.9 Cache-Control 101
14.9.1 What is Cachable 103
14.9.2 What May be Stored by Caches 103
14.9.3 Modifications of the Basic Expiration Mechanism 104
14.9.4 Cache Revalidation and Reload Controls 105
14.9.5 No-Transform Directive 107
14.9.6 Cache Control Extensions 108
14.10 Connection 109
14.11 Content-Base 109
14.12 Content-Encoding 110
14.13 Content-Language 110
14.14 Content-Length 111
14.15 Content-Location 112
14.16 Content-MD5 113
14.17 Content-Range 114
14.18 Content-Type 116
14.19 Date 116
14.20 ETag 117
14.21 Expires 117
14.22 From 118
14.23 Host 119
14.24 If-Modified-Since 119
14.25 If-Match 121
14.26 If-None-Match 122
14.27 If-Range 123
14.28 If-Unmodified-Since 124
14.29 Last-Modified 124
14.30 Location 125
14.31 Max-Forwards 125
14.32 Pragma 126
14.33 Proxy-Authenticate 127
14.34 Proxy-Authorization 127
14.35 Public 127
14.36 Range 128
14.36.1 Byte Ranges 128
14.36.2 Range Retrieval Requests 130
14.37 Referer 131
14.38 Retry-After 131
14.39 Server 132
14.40 Transfer-Encoding 132
14.41 Upgrade 132

14.42 User-Agent 134
14.43 Vary 134
14.44 Via 135
14.45 Warning 137
14.46 WWW-Authenticate 139
15 Security Considerations.....139
15.1 Authentication of Clients 139
15.2 Offering a Choice of Authentication Schemes 140
15.3 Abuse of Server Log Information 141
15.4 Transfer of Sensitive Information 141
15.5 Attacks Based On File and Path Names 142
15.6 Personal Information 143
15.7 Privacy Issues Connected to Accept Headers 143
15.8 DNS Spoofing 144
15.9 Location Headers and Spoofing 144
16 Acknowledgments.....144
17 References.....146
18 Authors' Addresses.....149
19 Appendices.....150
19.1 Internet Media Type message/http 150
19.2 Internet Media Type multipart/byteranges 150
19.3 Tolerant Applications 151
19.4 Differences Between HTTP Entities and MIME Entities.....152
19.4.1 Conversion to Canonical Form 152
19.4.2 Conversion of Date Formats 153
19.4.3 Introduction of Content-Encoding 153
19.4.4 No Content-Transfer-Encoding 153
19.4.5 HTTP Header Fields in Multipart Body-Parts 153
19.4.6 Introduction of Transfer-Encoding 154
19.4.7 MIME-Version 154
19.5 Changes from HTTP/1.0 154
19.5.1 Changes to Simplify Multi-homed Web Servers and Conserve IP Addresses 155
19.6 Additional Features 156
19.6.1 Additional Request Methods 156
19.6.2 Additional Header Field Definitions 156
19.7 Compatibility with Previous Versions 160
19.7.1 Compatibility with HTTP/1.0 Persistent Connections.....161

1 Introduction

1.1 Purpose

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World-Wide Web global information initiative since 1990. The first version of HTTP, referred to as HTTP/0.9, was a simple protocol for raw data transfer across the Internet. HTTP/1.0, as defined by RFC 1945 [6], improved the protocol by allowing messages to be in the format of MIME-like messages, containing metainformation about the data transferred and modifiers on the request/response semantics. However, HTTP/1.0 does not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections, and virtual hosts. In addition, the proliferation of incompletely-implemented applications calling themselves "HTTP/1.0" has necessitated a protocol version change in order for two communicating applications to determine each other's true capabilities.

This specification defines the protocol referred to as "HTTP/1.1". This protocol includes more stringent requirements than HTTP/1.0 in order to ensure reliable implementation of its features.

Practical information systems require more functionality than simple retrieval, including search, front-end update, and annotation. HTTP allows an open-ended set of methods that indicate the purpose of a request. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI) [3][20], as a location (URL) [4] or name (URN) , for indicating the resource to which a method is to be applied. Messages are passed in a format similar to that used by Internet mail as defined by the Multipurpose Internet Mail Extensions (MIME).

HTTP is also used as a generic protocol for communication between user agents and proxies/gateways to other Internet systems, including those supported by the SMTP [16], NNTP [13], FTP [18], Gopher [2], and WAIS [10] protocols. In this way, HTTP allows basic hypermedia access to resources available from diverse applications.

1.2 Requirements

This specification uses the same words as RFC 1123 [8] for defining the significance of each particular requirement. These words are:

MUST
: This word or the adjective "required" means that the item is an absolute requirement of the specification.

SHOULD
: This word or the adjective "recommended" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.

MAY
: This word or the adjective "optional" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

An implementation is not compliant if it fails to satisfy one or more of the MUST requirements for the protocols it implements. An implementation that satisfies all the MUST and all the SHOULD requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST requirements but not all the SHOULD requirements for its protocols is said to be "conditionally compliant."

1.3 Terminology

This specification uses a number of terms to refer to the roles played by participants in, and objects of, the HTTP communication.

connection
: A transport layer virtual circuit established between two programs for the purpose of communication.

message
: The basic unit of HTTP communication, consisting of a structured sequence of octets matching the syntax defined in section 4 and transmitted via the connection.

request
: An HTTP request message, as defined in section 5.

response
: An HTTP response message, as defined in section 6.

resource
: A network data object or service that can be identified by a URI, as defined in section 3.2. Resources may be available in multiple representations (e.g. multiple languages, data formats, size, resolutions) or vary in other ways.

entity
The information transferred as the payload of a request or response. An entity consists of metainformation in the form of entity-header fields and content in the form of an entity-body, as described in section 7.

representation
An entity included with a response that is subject to content negotiation, as described in section 12. There may exist multiple representations associated with a particular response status.

content negotiation
The mechanism for selecting the appropriate representation when servicing a request, as described in section 12. The representation of entities in any response can be negotiated (including error responses).

variant
A resource may have one, or more than one, representation(s) associated with it at any given instant. Each of these representations is termed a `variant.' Use of the term `variant' does not necessarily imply that the resource is subject to content negotiation.

client
A program that establishes connections for the purpose of sending requests.

user agent
The client which initiates a request. These are often browsers, editors, spiders (web-traversing robots), or other end user tools.

server
An application program that accepts connections in order to service requests by sending back responses. Any given program may be capable of being both a client and a server; our use of these terms refers only to the role being performed by the program for a particular connection, rather than to the program's capabilities in general. Likewise, any server may act as an origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each request.

origin server
The server on which a given resource resides or is to be created.

proxy
An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, with possible translation, to other servers. A proxy must implement both the client and server requirements of this specification.

gateway
A server which acts as an intermediary for some other server. Unlike a proxy, a gateway receives requests as if it were the origin server for the requested resource; the requesting client may not be aware that it is communicating with a gateway.

tunnel
An intermediary program which is acting as a blind relay between two connections. Once active, a tunnel is not considered a party to the HTTP communication, though the tunnel may have been initiated by an HTTP request. The tunnel ceases to exist when both ends of the relayed connections are closed.

cache
A program's local store of response messages and the subsystem that controls its message storage, retrieval, and deletion. A cache stores cachable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests. Any client or server may include a cache, though a cache cannot be used by a server that is acting as a tunnel.

cachable
A response is cachable if a cache is allowed to store a copy of the response message for use in answering subsequent requests. The rules for determining the cachability of HTTP responses are defined in section 13. Even if a resource is cachable, there may be additional constraints on whether a cache can use the cached copy for a particular request.

first-hand
A response is first-hand if it comes directly and without unnecessary delay from the origin server, perhaps via one or more proxies. A response is also first-hand if its validity has just been checked directly with the origin server.

explicit expiration time
The time at which the origin server intends that an entity should no longer be returned by a cache without further validation.

heuristic expiration time
An expiration time assigned by a cache when no explicit expiration time is available.

age
The age of a response is the time since it was sent by, or successfully validated with, the origin server.

freshness lifetime
The length of time between the generation of a response and its expiration time.

fresh
A response is fresh if its age has not yet exceeded its freshness lifetime.

stale
A response is stale if its age has passed its freshness lifetime.

semantically transparent
A cache behaves in a "semantically transparent" manner, with respect to a particular response, when its use affects neither the requesting client nor the origin server, except to improve performance. When a cache is semantically transparent, the client receives exactly the same response (except for hop-by-hop headers) that it would have received had its request been handled directly by the origin server.

validator
A protocol element (e.g., an entity tag or a Last-Modified time) that is used to find out whether a cache entry is an equivalent copy of an entity.

1.4 Overall Operation

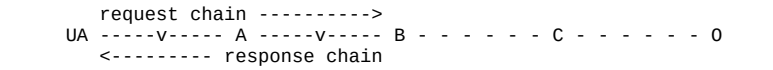
The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible entity-body content. The relationship between HTTP and MIME is described in appendix 19.4.

Most HTTP communication is initiated by a user agent and consists of a request to be applied to a resource on some origin server. In the simplest case, this may be accomplished via a single connection (v) between the user agent (UA) and the origin server (O).

A more complicated situation occurs when one or more intermediaries are present in the request/response chain. There are three common forms of intermediary: proxy, gateway, and tunnel. A proxy is a forwarding agent, receiving requests for a URI in its absolute form, rewriting all or part of the message, and forwarding the reformatted request toward the server identified by the URI. A gateway is a receiving agent, acting as a layer above some other server(s) and, if necessary, translating the requests to the underlying server's protocol. A tunnel acts as a relay point between two connections without changing the messages; tunnels are used when the communication needs to pass through an intermediary (such as a firewall) even when the intermediary cannot understand the contents of the messages.

The figure above shows three intermediaries (A, B, and C) between the user agent and origin server. A request or response message that travels the whole chain will pass through four separate connections. This distinction is important because some HTTP communication options may apply only to the connection with the nearest, non-tunnel neighbor, only to the end-points of the chain, or to all connections along the chain. Although the diagram is linear, each participant may be engaged in multiple, simultaneous communications. For example, B may be receiving requests from many clients other than A, and/or forwarding requests to servers other than C, at the same time that it is handling A's request.

Any party to the communication which is not acting as a tunnel may employ an internal cache for handling requests. The effect of a cache is that the request/response chain is shortened if one of the participants along the chain has a cached response applicable to that request. The following illustrates the resulting chain if B has a cached copy of an earlier response from O (via C) for a request which has not been cached by UA or A.



Not all responses are usefully cachable, and some requests may contain modifiers which place special requirements on cache behavior. HTTP requirements for cache behavior and cachable responses are defined in section 13.

In fact, there are a wide variety of architectures and configurations of caches and proxies currently being experimented with or deployed across the World Wide Web; these systems include national hierarchies of proxy caches to save transoceanic bandwidth, systems that broadcast or multicast cache entries, organizations that distribute subsets of cached data via CD-ROM, and so on. HTTP systems are used in corporate intranets over high-bandwidth links, and for access via PDAs with low-power radio links and intermittent connectivity. The goal of HTTP/1.1 is to support the wide diversity of configurations already deployed while introducing protocol constructs that meet the needs of those who build web applications that require high reliability and, failing that, at least reliable indications of failure.

HTTP communication usually takes place over TCP/IP connections. The default port is TCP 80, but other ports can be used. This does not preclude HTTP from being implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used; the mapping of the HTTP/1.1 request and response structures onto the transport data units of the protocol in question is outside the scope of this specification.

In HTTP/1.0, most implementations used a new connection for each request/response exchange. In HTTP/1.1, a connection may be used for one or more request/response exchanges, although connections may be closed for a variety of reasons (see section 8.1).

2 Notational Conventions and Generic Grammar

2.1 Augmented BNF

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) similar to that used by RFC 822 [9]. Implementers will need to be familiar with the notation in order to understand this specification. The augmented BNF includes the following constructs:

name = definition
: The name of a rule is simply the name itself (without any enclosing "<" and ">") and is separated from its definition by the equal "=" character. Whitespace is only significant in that indentation of continuation lines is used to indicate a rule definition that spans more than one line. Certain basic rules are in uppercase, such as SP, LWS, HT, CRLF, DIGIT, ALPHA, etc. Angle brackets are used within definitions whenever their presence will facilitate discerning the use of rule names.

"literal"
: Quotation marks surround literal text. Unless stated otherwise, the text is case-insensitive.

rule1 | rule2
: Elements separated by a bar ("|") are alternatives, e.g., "yes | no" will accept yes or no.

(rule1 rule2)
: Elements enclosed in parentheses are treated as a single element. Thus, "(elem (foo | bar) elem)" allows the token sequences "elem foo elem" and "elem bar elem".

*rule
: The character "*" preceding an element indicates repetition. The full form is "<n>*<m>element" indicating at least <n> and at most <m> occurrences of element. Default values are 0 and infinity so that "*(element)" allows any number, including zero; "1*element" requires at least one; and "1*2element" allows one or two.

[rule]
: Square brackets enclose optional elements; "[foo bar]" is equivalent to "*1(foo bar)".

N rule
: Specific repetition: "<n>(element)" is equivalent to "<n>*<n>(element)"; that is, exactly <n> occurrences of (element). Thus 2DIGIT is a 2-digit number, and 3ALPHA is a string of three alphabetic characters.

#rule
: A construct "#" is defined, similar to "*", for defining lists of elements. The full form is "<n>#<m>element " indicating at least <n> and at most <m> elements, each separated by one or more commas (",") and optional linear whitespace (LWS). This makes the usual form of lists very easy; a rule such as "(*LWS element *(*LWS "," *LWS element)) " can be shown as "1#element". Wherever this construct is used, null elements are allowed, but do not contribute

to the count of elements present. That is, "(element), , (element) " is permitted, but counts as only two elements. Therefore, where at least one element is required, at least one non-null element must be present. Default values are 0 and infinity so that "#element" allows any number, including zero; "1#element" requires at least one; and "1#2element" allows one or two.

; comment
A semi-colon, set off some distance to the right of rule text, starts a comment that continues to the end of line. This is a simple way of including useful notes in parallel with the specifications.

implied *LWS
The grammar described by this specification is word-based. Except where noted otherwise, linear whitespace (LWS) can be included between any two adjacent words (token or quoted-string), and between adjacent tokens and delimiters (tspecials), without changing the interpretation of a field. At least one delimiter (tspecials) must exist between any two tokens, since they would otherwise be interpreted as a single token.

2.2 Basic Rules

The following rules are used throughout this specification to describe basic parsing constructs. The US-ASCII coded character set is defined by ANSI X3.4-1986 [21].

```
OCTET          = <any 8-bit sequence of data>
CHAR           = <any US-ASCII character (octets 0 - 127)>
UPALPHA        = <any US-ASCII uppercase letter "A.."Z">
LOALPHA        = <any US-ASCII lowercase letter "a.."z">
ALPHA          = UPALPHA | LOALPHA
DIGIT          = <any US-ASCII digit "0.."9">
CTL            = <any US-ASCII control character
                  (octets 0 - 31) and DEL (127)>
CR             = <US-ASCII CR, carriage return (13)>
LF             = <US-ASCII LF, linefeed (10)>
SP             = <US-ASCII SP, space (32)>
HT             = <US-ASCII HT, horizontal-tab (9)>
<">            = <US-ASCII double-quote mark (34)>
```


HTTP/1.1 defines the sequence CR LF as the end-of-line marker for all protocol elements except the entity-body (see appendix 19.3 for tolerant applications). The end-of-line marker within an entity-body is defined by its associated media type, as described in section 3.7.

```
CRLF           = CR LF
```

HTTP/1.1 headers can be folded onto multiple lines if the continuation line begins with a space or horizontal tab. All linear white space, including folding, has the same semantics as SP.

```
LWS            = [CRLF] 1*( SP | HT )
```

The TEXT rule is only used for descriptive field contents and values that are not intended to be interpreted by the message parser. Words of *TEXT may contain characters from character sets other than ISO 8859-1 [22] only when encoded according to the rules of RFC 1522 [14].

```
TEXT           = <any OCTET except CTLs,
                  but including LWS>
```

Hexadecimal numeric characters are used in several protocol elements.

```
HEX            = "A" | "B" | "C" | "D" | "E" | "F"
               | "a" | "b" | "c" | "d" | "e" | "f" | DIGIT
```

Many HTTP/1.1 header field values consist of words separated by LWS or special characters. These special characters MUST be in a quoted string to be used within a parameter value.

```
token          = 1*<any CHAR except CTLs or tspecials>

tspecials      = "(" | ")" | "<" | ">" | "@"
               | "," | ";" | ":" | "\" | <">
               | "/" | "[" | "]" | "?" | "="
               | "{" | "}" | SP | HT
```

Comments can be included in some HTTP header fields by surrounding the comment text with parentheses. Comments are only allowed in fields containing "comment" as part of their field value definition. In all other fields, parentheses are considered part of the field value.

```
comment        = "(" *( ctext | comment ) ")"
ctext          = <any TEXT excluding "(" and ")">
```

A string of text is parsed as a single word if it is quoted using double-quote marks.

```
quoted-string  = ( <"> *(qdtext) <"> )

qdtext         = <any TEXT except <">>
```

The backslash character ("\") may be used as a single-character quoting mechanism only within quoted-string and comment constructs.

```
quoted-pair    = "\" CHAR
```

3 Protocol Parameters

3.1 HTTP Version

HTTP uses a "<major>.<minor>" numbering scheme to indicate versions of the protocol. The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further HTTP communication, rather than the features obtained via that communication. No change is made to the version number for the addition of message components which do not affect communication behavior or which only add to extensible field values. The <minor> number is incremented when the changes made to the protocol add features which do not change the general message parsing algorithm, but which may add to the message semantics and imply additional capabilities of the sender. The <major> number is incremented when the format of a message within the protocol is changed.

The version of an HTTP message is indicated by an HTTP-Version field in the first line of the message.

```
HTTP-Version   = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

Note that the major and minor numbers MUST be treated as separate integers and that each may be incremented higher than a single digit. Thus, HTTP/2.4 is a lower version than HTTP/2.13, which in turn is lower than HTTP/12.3. Leading zeros MUST be ignored by recipients and MUST NOT be sent.

Applications sending Request or Response messages, as defined by this specification, MUST include an HTTP-Version of "HTTP/1.1". Use of this version number indicates that the sending application is at least conditionally compliant with this specification.

The HTTP version of an application is the highest HTTP version for which the application is at least conditionally compliant.

Proxy and gateway applications must be careful when forwarding messages in protocol versions different from that of the application. Since the protocol version indicates the protocol capability of the sender, a proxy/gateway MUST never send a message with a version indicator which is greater than its actual version; if a higher version request is received, the proxy/gateway MUST either downgrade the request version, respond with an error, or switch to tunnel behavior. Requests with a version lower than that of the proxy/gateway's version MAY be upgraded before being forwarded; the proxy/gateway's response to that request MUST be in the same major version as the request.

> Note: Converting between versions of HTTP may involve modification of header fields required or forbidden by the versions involved.

3.2 Uniform Resource Identifiers

URIs have been known by many names: WWW addresses, Universal Document Identifiers, Universal Resource Identifiers , and finally the combination of Uniform Resource Locators (URL) and Names (URN). As far as HTTP is concerned, Uniform Resource Identifiers are simply formatted strings which identify--via name, location, or any other characteristic--a resource.

3.2.1 General Syntax

URIs in HTTP can be represented in absolute form or relative to some known base URI, depending upon the context of their use. The two forms are differentiated by the fact that absolute URIs always begin with a scheme name followed by a colon.

```
URI            = ( absoluteURI | relativeURI ) [ "#" fragment ]

absoluteURI    = scheme ":" *( uchar | reserved )

relativeURI    = net_path | abs_path | rel_path

net_path       = "//" net_loc [ abs_path ]
abs_path       = "/" rel_path
rel_path       = [ path ] [ ";" params ] [ "?" query ]

path           = fsegment *( "/" segment )
fsegment       = 1*pchar
segment        = *pchar

params         = param *( ";" param )
param          = *( pchar | "/" )
```

```
scheme          = 1*( ALPHA | DIGIT | "+" | "-" | "." )
net_loc         = *( pchar | ";" | "?" )

query           = *( uchar | reserved )
fragment        = *( uchar | reserved )

pchar           = uchar | ":" | "@" | "&" | "=" | "+"
uchar           = unreserved | escape
unreserved      = ALPHA | DIGIT | safe | extra | national

escape          = "%" HEX HEX
reserved        = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+"
extra           = "!" | "*" | "'" | "(" | ")" | ","
safe            = "$" | "-" | "_" | "."
unsafe          = CTL | SP | <"> | "#" | "%" | "<" | ">"
national        = <any OCTET excluding ALPHA, DIGIT,
                  reserved, extra, safe, and unsafe>
```

For definitive information on URL syntax and semantics, see RFC 1738 [4] and RFC 1808 [11]. The BNF above includes national characters not allowed in valid URLs as specified by RFC 1738, since HTTP servers are not restricted in the set of unreserved characters allowed to represent the rel_path part of addresses, and HTTP proxies may receive requests for URIs not defined by RFC 1738.

The HTTP protocol does not place any a priori limit on the length of a URI. Servers MUST be able to handle the URI of any resource they serve, and SHOULD be able to handle URIs of unbounded length if they provide GET-based forms that could generate such URIs. A server SHOULD return 414 (Request-URI Too Long) status if a URI is longer than the server can handle (see section 10.4.15).

> Note: Servers should be cautious about depending on URI lengths above 255 bytes, because some older client or proxy implementations may not properly support these lengths.

3.2.2 http URL

The "http" scheme is used to locate network resources via the HTTP protocol. This section defines the scheme-specific syntax and semantics for http URLs.


```
http_URL        = "http:" "//" host [ ":" port ] [ abs_path ]

host            = <A legal Internet host domain name
                   or IP address (in dotted-decimal form),
                   as defined by Section 2.1 of RFC 1123>

port            = *DIGIT
```

If the port is empty or not given, port 80 is assumed. The semantics are that the identified resource is located at the server listening for TCP connections on that port of that host, and the Request-URI for the resource is abs_path. The use of IP addresses in URL's SHOULD be avoided whenever possible (see RFC 1900 [24]). If the abs_path is not present in the URL, it MUST be given as "/" when used as a Request-URI for a resource (section 5.1.2).

3.2.3 URI Comparison

When comparing two URIs to decide if they match or not, a client SHOULD use a case-sensitive octet-by-octet comparison of the entire URIs, with these exceptions:

- o A port that is empty or not given is equivalent to the default port for that URI;
- o Comparisons of host names MUST be case-insensitive;
- o Comparisons of scheme names MUST be case-insensitive;
- o An empty abs_path is equivalent to an abs_path of "/".

Characters other than those in the "reserved" and "unsafe" sets (see section 3.2) are equivalent to their ""%" HEX HEX" encodings.

For example, the following three URIs are equivalent:

```
http://abc.com:80/~smith/home.html
http://ABC.com/%7Esmith/home.html
http://ABC.com:/%7esmith/home.html
```

3.3 Date/Time Formats

3.3.1 Full Date

HTTP applications have historically allowed three different formats for the representation of date/time stamps:

```
Sun, 06 Nov 1994 08:49:37 GMT  ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
Sun Nov  6 08:49:37 1994       ; ANSI C's asctime() format
```

The first format is preferred as an Internet standard and represents a fixed-length subset of that defined by RFC 1123 (an update to RFC 822). The second format is in common use, but is based on the obsolete RFC 850 [12] date format and lacks a four-digit year. HTTP/1.1 clients and servers that parse the date value MUST accept all three formats (for compatibility with HTTP/1.0), though they MUST only generate the RFC 1123 format for representing HTTP-date values in header fields.

> Note: Recipients of date values are encouraged to be robust in accepting date values that may have been sent by non-HTTP applications, as is sometimes the case when retrieving or posting messages via proxies/gateways to SMTP or NNTP.

All HTTP date/time stamps MUST be represented in Greenwich Mean Time (GMT), without exception. This is indicated in the first two formats by the inclusion of "GMT" as the three-letter abbreviation for time zone, and MUST be assumed when reading the asctime format.

```
HTTP-date    = rfc1123-date | rfc850-date | asctime-date

rfc1123-date = wkday "," SP date1 SP time SP "GMT"
rfc850-date  = weekday "," SP date2 SP time SP "GMT"
asctime-date = wkday SP date3 SP time SP 4DIGIT

date1        = 2DIGIT SP month SP 4DIGIT
               ; day month year (e.g., 02 Jun 1982)
date2        = 2DIGIT "-" month "-" 2DIGIT
               ; day-month-year (e.g., 02-Jun-82)
date3        = month SP ( 2DIGIT | ( SP 1DIGIT ))
               ; month day (e.g., Jun  2)

time         = 2DIGIT ":" 2DIGIT ":" 2DIGIT
               ; 00:00:00 - 23:59:59

wkday        = "Mon" | "Tue" | "Wed"
             | "Thu" | "Fri" | "Sat" | "Sun"
```



```
weekday      = "Monday" | "Tuesday" | "Wednesday"
             | "Thursday" | "Friday" | "Saturday" | "Sunday"

month        = "Jan" | "Feb" | "Mar" | "Apr"
             | "May" | "Jun" | "Jul" | "Aug"
             | "Sep" | "Oct" | "Nov" | "Dec"
```

> Note: HTTP requirements for the date/time stamp format apply only to their usage within the protocol stream. Clients and servers are not required to use these formats for user presentation, request logging, etc.

3.3.2 Delta Seconds

Some HTTP header fields allow a time value to be specified as an integer number of seconds, represented in decimal, after the time that the message was received.

```
delta-seconds  = 1*DIGIT
```

3.4 Character Sets

HTTP uses the same definition of the term "character set" as that described for MIME:

> The term "character set" is used in this document to refer to a method used with one or more tables to convert a sequence of octets into a sequence of characters. Note that unconditional conversion in the other direction is not required, in that not all characters may be available in a given character set and a character set may provide more than one sequence of octets to represent a particular character. This definition is intended to allow various kinds of character encodings, from simple single-table mappings such as US-ASCII to complex table switching methods such as those that use ISO 2022's techniques. However, the definition associated with a MIME character set name MUST fully specify the mapping to be performed from octets to characters. In particular, use of external profiling information to determine the exact mapping is not permitted.
>
> Note: This use of the term "character set" is more commonly referred to as a "character encoding." However, since HTTP and MIME share the same registry, it is important that the terminology also be shared.

HTTP character sets are identified by case-insensitive tokens. The complete set of tokens is defined by the IANA Character Set registry [19].

```
charset = token
```

Although HTTP allows an arbitrary token to be used as a charset value, any token that has a predefined value within the IANA Character Set registry MUST represent the character set defined by that registry. Applications SHOULD limit their use of character sets to those defined by the IANA registry.

3.5 Content Codings

Content coding values indicate an encoding transformation that has been or can be applied to an entity. Content codings are primarily used to allow a document to be compressed or otherwise usefully transformed without losing the identity of its underlying media type and without loss of information. Frequently, the entity is stored in coded form, transmitted directly, and only decoded by the recipient.

```
content-coding   = token
```

All content-coding values are case-insensitive. HTTP/1.1 uses content-coding values in the Accept-Encoding (section 14.3) and Content-Encoding (section 14.12) header fields. Although the value describes the content-coding, what is more important is that it indicates what decoding mechanism will be required to remove the encoding.

The Internet Assigned Numbers Authority (IANA) acts as a registry for content-coding value tokens. Initially, the registry contains the following tokens:

gzip An encoding format produced by the file compression program "gzip" (GNU zip) as described in RFC 1952 [25]. This format is a Lempel-Ziv coding (LZ77) with a 32 bit CRC.

compress
The encoding format produced by the common UNIX file compression program "compress". This format is an adaptive Lempel-Ziv-Welch coding (LZW).

Note: Use of program names for the identification of encoding formats is not desirable and should be discouraged for future encodings. Their use here is representative of historical practice, not good design. For compatibility with previous implementations of HTTP, applications should consider "x-gzip" and "x-compress" to be equivalent to "gzip" and "compress" respectively.

deflate The "zlib" format defined in RFC 1950[31] in combination with the "deflate" compression mechanism described in RFC 1951[29].

New content-coding value tokens should be registered; to allow interoperability between clients and servers, specifications of the content coding algorithms needed to implement a new value should be publicly available and adequate for independent implementation, and conform to the purpose of content coding defined in this section.

3.6 Transfer Codings

Transfer coding values are used to indicate an encoding transformation that has been, can be, or may need to be applied to an entity-body in order to ensure "safe transport" through the network. This differs from a content coding in that the transfer coding is a property of the message, not of the original entity.

```
transfer-coding         = "chunked" | transfer-extension

transfer-extension      = token
```

All transfer-coding values are case-insensitive. HTTP/1.1 uses transfer coding values in the Transfer-Encoding header field (section 14.40).

Transfer codings are analogous to the Content-Transfer-Encoding values of MIME , which were designed to enable safe transport of binary data over a 7-bit transport service. However, safe transport has a different focus for an 8bit-clean transfer protocol. In HTTP, the only unsafe characteristic of message-bodies is the difficulty in determining the exact body length (section 7.2.2), or the desire to encrypt data over a shared transport.

The chunked encoding modifies the body of a message in order to transfer it as a series of chunks, each with its own size indicator, followed by an optional footer containing entity-header fields. This allows dynamically-produced content to be transferred along with the information necessary for the recipient to verify that it has received the full message.

```
Chunked-Body   = *chunk
                 "0" CRLF
                 footer
                 CRLF

chunk          = chunk-size [ chunk-ext ] CRLF
                 chunk-data CRLF

hex-no-zero    = <HEX excluding "0">

chunk-size     = hex-no-zero *HEX
chunk-ext      = *( ";" chunk-ext-name [ "=" chunk-ext-value ] )
chunk-ext-name = token
chunk-ext-val  = token | quoted-string
chunk-data     = chunk-size(OCTET)

footer         = *entity-header
```

The chunked encoding is ended by a zero-sized chunk followed by the footer, which is terminated by an empty line. The purpose of the footer is to provide an efficient way to supply information about an entity that is generated dynamically; applications MUST NOT send header fields in the footer which are not explicitly defined as being appropriate for the footer, such as Content-MD5 or future extensions to HTTP for digital signatures or other facilities.

An example process for decoding a Chunked-Body is presented in appendix 19.4.6.

All HTTP/1.1 applications MUST be able to receive and decode the "chunked" transfer coding, and MUST ignore transfer coding extensions they do not understand. A server which receives an entity-body with a transfer-coding it does not understand SHOULD return 501 (Unimplemented), and close the connection. A server MUST NOT send transfer-codings to an HTTP/1.0 client.

3.7 Media Types

HTTP uses Internet Media Types in the Content-Type (section 14.18) and Accept (section 14.1) header fields in order to provide open and extensible data typing and type negotiation.

```
media-type     = type "/" subtype *( ";" parameter )
type           = token
subtype        = token
```

Parameters may follow the type/subtype in the form of attribute/value pairs.


```
parameter      = attribute "=" value
attribute      = token
value          = token | quoted-string
```

The type, subtype, and parameter attribute names are case-insensitive. Parameter values may or may not be case-sensitive, depending on the semantics of the parameter name. Linear white space (LWS) MUST NOT be used between the type and subtype, nor between an attribute and its value. User agents that recognize the media-type MUST process (or arrange to be processed by any external applications used to process that type/subtype by the user agent) the parameters for that MIME type as described by that type/subtype definition to the and inform the user of any problems discovered.

> Note: some older HTTP applications do not recognize media type parameters. When sending data to older HTTP applications, implementations should only use media type parameters when they are required by that type/subtype definition.

Media-type values are registered with the Internet Assigned Number Authority (IANA). The media type registration process is outlined in RFC 2048 [17]. Use of non-registered media types is discouraged.

3.7.1 Canonicalization and Text Defaults

Internet media types are registered with a canonical form. In general, an entity-body transferred via HTTP messages MUST be represented in the appropriate canonical form prior to its transmission; the exception is "text" types, as defined in the next paragraph.

When in canonical form, media subtypes of the "text" type use CRLF as the text line break. HTTP relaxes this requirement and allows the transport of text media with plain CR or LF alone representing a line break when it is done consistently for an entire entity-body. HTTP applications MUST accept CRLF, bare CR, and bare LF as being representative of a line break in text media received via HTTP. In addition, if the text is represented in a character set that does not use octets 13 and 10 for CR and LF respectively, as is the case for some multi-byte character sets, HTTP allows the use of whatever octet sequences are defined by that character set to represent the equivalent of CR and LF for line breaks. This flexibility regarding line breaks applies only to text media in the entity-body; a bare CR or LF MUST NOT be substituted for CRLF within any of the HTTP control structures (such as header fields and multipart boundaries).

If an entity-body is encoded with a Content-Encoding, the underlying data MUST be in a form defined above prior to being encoded.

The "charset" parameter is used with some media types to define the character set (section 3.4) of the data. When no explicit charset parameter is provided by the sender, media subtypes of the "text" type are defined to have a default charset value of "ISO-8859-1" when received via HTTP. Data in character sets other than "ISO-8859-1" or its subsets MUST be labeled with an appropriate charset value.

Some HTTP/1.0 software has interpreted a Content-Type header without charset parameter incorrectly to mean "recipient should guess." Senders wishing to defeat this behavior MAY include a charset parameter even when the charset is ISO-8859-1 and SHOULD do so when it is known that it will not confuse the recipient.

Unfortunately, some older HTTP/1.0 clients did not deal properly with an explicit charset parameter. HTTP/1.1 recipients MUST respect the charset label provided by the sender; and those user agents that have a provision to "guess" a charset MUST use the charset from the content-type field if they support that charset, rather than the recipient's preference, when initially displaying a document.

3.7.2 Multipart Types

MIME provides for a number of "multipart" types -- encapsulations of one or more entities within a single message-body. All multipart types share a common syntax, as defined in MIME [7], and MUST include a boundary parameter as part of the media type value. The message body is itself a protocol element and MUST therefore use only CRLF to represent line breaks between body-parts. Unlike in MIME, the epilogue of any multipart message MUST be empty; HTTP applications MUST NOT transmit the epilogue (even if the original multipart contains an epilogue).

In HTTP, multipart body-parts MAY contain header fields which are significant to the meaning of that part. A Content-Location header field (section 14.15) SHOULD be included in the body-part of each enclosed entity that can be identified by a URL.

In general, an HTTP user agent SHOULD follow the same or similar behavior as a MIME user agent would upon receipt of a multipart type. If an application receives an unrecognized multipart subtype, the application MUST treat it as being equivalent to "multipart/mixed".

> Note: The "multipart/form-data" type has been specifically defined for carrying form data suitable for processing via the POST request method, as described in RFC 1867 [15].

3.8 Product Tokens

Product tokens are used to allow communicating applications to identify themselves by software name and version. Most fields using product tokens also allow sub-products which form a significant part of the application to be listed, separated by whitespace. By convention, the products are listed in order of their significance for identifying the application.

```
product         = token ["/" product-version]
product-version = token
```

Examples:

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
Server: Apache/0.8.4
```

Product tokens should be short and to the point -- use of them for advertising or other non-essential information is explicitly forbidden. Although any token character may appear in a product-version, this token SHOULD only be used for a version identifier (i.e., successive versions of the same product SHOULD only differ in the product-version portion of the product value).

3.9 Quality Values

HTTP content negotiation (section 12) uses short "floating point" numbers to indicate the relative importance ("weight") of various negotiable parameters. A weight is normalized to a real number in the range 0 through 1, where 0 is the minimum and 1 the maximum value. HTTP/1.1 applications MUST NOT generate more than three digits after the decimal point. User configuration of these values SHOULD also be limited in this fashion.

```
qvalue         = ( "0" [ "." 0*3DIGIT ] )
               | ( "1" [ "." 0*3("0") ] )
```

"Quality values" is a misnomer, since these values merely represent relative degradation in desired quality.

3.10 Language Tags

A language tag identifies a natural language spoken, written, or otherwise conveyed by human beings for communication of information to other human beings. Computer languages are explicitly excluded. HTTP uses language tags within the Accept-Language and Content-Language fields.

The syntax and registry of HTTP language tags is the same as that defined by RFC 1766 [1]. In summary, a language tag is composed of 1 or more parts: A primary language tag and a possibly empty series of subtags:

```
        language-tag  = primary-tag *( "-" subtag )

        primary-tag   = 1*8ALPHA
        subtag        = 1*8ALPHA
```

Whitespace is not allowed within the tag and all tags are case-insensitive. The name space of language tags is administered by the IANA. Example tags include:

```
       en, en-US, en-cockney, i-cherokee, x-pig-latin
```

where any two-letter primary-tag is an ISO 639 language abbreviation and any two-letter initial subtag is an ISO 3166 country code. (The last three tags above are not registered tags; all but the last are examples of tags which could be registered in future.)

3.11 Entity Tags

Entity tags are used for comparing two or more entities from the same requested resource. HTTP/1.1 uses entity tags in the ETag (section 14.20), If-Match (section 14.25), If-None-Match (section 14.26), and If-Range (section 14.27) header fields. The definition of how they are used and compared as cache validators is in section 13.3.3. An entity tag consists of an opaque quoted string, possibly prefixed by a weakness indicator.

```
      entity-tag = [ weak ] opaque-tag

      weak       = "W/"
      opaque-tag = quoted-string
```

A "strong entity tag" may be shared by two entities of a resource only if they are equivalent by octet equality.

A "weak entity tag," indicated by the "W/" prefix, may be shared by two entities of a resource only if the entities are equivalent and could be substituted for each other with no significant change in semantics. A weak entity tag can only be used for weak comparison.

An entity tag MUST be unique across all versions of all entities associated with a particular resource. A given entity tag value may be used for entities obtained by requests on different URIs without implying anything about the equivalence of those entities.

3.12 Range Units

HTTP/1.1 allows a client to request that only part (a range of) the response entity be included within the response. HTTP/1.1 uses range units in the Range (section 14.36) and Content-Range (section 14.17) header fields. An entity may be broken down into subranges according to various structural units.

```
      range-unit       = bytes-unit | other-range-unit

      bytes-unit       = "bytes"
      other-range-unit = token
```

The only range unit defined by HTTP/1.1 is "bytes". HTTP/1.1 implementations may ignore ranges specified using other units. HTTP/1.1 has been designed to allow implementations of applications that do not depend on knowledge of ranges.

4 HTTP Message

4.1 Message Types

HTTP messages consist of requests from client to server and responses from server to client.

```
       HTTP-message   = Request | Response     ; HTTP/1.1 messages
```

Request (section 5) and Response (section 6) messages use the generic message format of RFC 822 [9] for transferring entities (the payload of the message). Both types of message consist of a start-line, one or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields, and an optional message-body.

```
        generic-message = start-line
                          *message-header
                          CRLF
                          [ message-body ]

        start-line      = Request-Line | Status-Line
```

In the interest of robustness, servers SHOULD ignore any empty line(s) received where a Request-Line is expected. In other words, if the server is reading the protocol stream at the beginning of a message and receives a CRLF first, it should ignore the CRLF.

Note: certain buggy HTTP/1.0 client implementations generate an extra CRLF's after a POST request. To restate what is explicitly forbidden by the BNF, an HTTP/1.1 client must not preface or follow a request with an extra CRLF.

4.2 Message Headers

HTTP header fields, which include general-header (section 4.5), request-header (section 5.3), response-header (section 6.2), and entity-header (section 7.1) fields, follow the same generic format as that given in Section 3.1 of RFC 822 [9]. Each header field consists of a name followed by a colon (":") and the field value. Field names are case-insensitive. The field value may be preceded by any amount of LWS, though a single SP is preferred. Header fields can be extended over multiple lines by preceding each extra line with at least one SP or HT. Applications SHOULD follow "common form" when generating HTTP constructs, since there might exist some implementations that fail to accept anything beyond the common forms.

```
message-header = field-name ":" [ field-value ] CRLF

field-name     = token
field-value    = *( field-content | LWS )

field-content  = <the OCTETs making up the field-value
                 and consisting of either *TEXT or combinations
                 of token, tspecials, and quoted-string>
```

The order in which header fields with differing field names are received is not significant. However, it is "good practice" to send general-header fields first, followed by request-header or response-header fields, and ending with the entity-header fields.

Multiple message-header fields with the same field-name may be present in a message if and only if the entire field-value for that header field is defined as a comma-separated list [i.e., #(values)]. It MUST be possible to combine the multiple header fields into one "field-name: field-value" pair, without changing the semantics of the message, by appending each subsequent field-value to the first, each separated by a comma. The order in which header fields with the same field-name are received is therefore significant to the interpretation of the combined field value, and thus a proxy MUST NOT change the order of these field values when a message is forwarded.

4.3 Message Body

The message-body (if any) of an HTTP message is used to carry the entity-body associated with the request or response. The message-body differs from the entity-body only when a transfer coding has been applied, as indicated by the Transfer-Encoding header field (section 14.40).

```
message-body = entity-body
             | <entity-body encoded as per Transfer-Encoding>
```

Transfer-Encoding MUST be used to indicate any transfer codings applied by an application to ensure safe and proper transfer of the message. Transfer-Encoding is a property of the message, not of the entity, and thus can be added or removed by any application along the request/response chain.

The rules for when a message-body is allowed in a message differ for requests and responses.

The presence of a message-body in a request is signaled by the inclusion of a Content-Length or Transfer-Encoding header field in the request's message-headers. A message-body MAY be included in a request only when the request method (section 5.1.1) allows an entity-body.

For response messages, whether or not a message-body is included with a message is dependent on both the request method and the response status code (section 6.1.1). All responses to the HEAD request method MUST NOT include a message-body, even though the presence of entity-header fields might lead one to believe they do. All 1xx (informational), 204 (no content), and 304 (not modified) responses MUST NOT include a message-body. All other responses do include a message-body, although it may be of zero length.

4.4 Message Length

When a message-body is included with a message, the length of that body is determined by one of the following (in order of precedence):

1. Any response message which MUST NOT include a message-body (such as the 1xx, 204, and 304 responses and any response to a HEAD request) is always terminated by the first empty line after the header fields, regardless of the entity-header fields present in the message.

2. If a Transfer-Encoding header field (section 14.40) is present and indicates that the "chunked" transfer coding has been applied, then

the length is defined by the chunked encoding (section 3.6).

3. If a Content-Length header field (section 14.14) is present, its value in bytes represents the length of the message-body.

4. If the message uses the media type "multipart/byteranges", which is self-delimiting, then that defines the length. This media type MUST NOT be used unless the sender knows that the recipient can parse it; the presence in a request of a Range header with multiple byte-range specifiers implies that the client can parse multipart/byteranges responses.

5. By the server closing the connection. (Closing the connection cannot be used to indicate the end of a request body, since that would leave no possibility for the server to send back a response.)

For compatibility with HTTP/1.0 applications, HTTP/1.1 requests containing a message-body MUST include a valid Content-Length header field unless the server is known to be HTTP/1.1 compliant. If a request contains a message-body and a Content-Length is not given, the server SHOULD respond with 400 (bad request) if it cannot determine the length of the message, or with 411 (length required) if it wishes to insist on receiving a valid Content-Length.

All HTTP/1.1 applications that receive entities MUST accept the "chunked" transfer coding (section 3.6), thus allowing this mechanism to be used for messages when the message length cannot be determined in advance.

Messages MUST NOT include both a Content-Length header field and the "chunked" transfer coding. If both are received, the Content-Length MUST be ignored.

When a Content-Length is given in a message where a message-body is allowed, its field value MUST exactly match the number of OCTETs in the message-body. HTTP/1.1 user agents MUST notify the user when an invalid length is received and detected.

4.5 General Header Fields

There are a few header fields which have general applicability for both request and response messages, but which do not apply to the entity being transferred. These header fields apply only to the message being transmitted.

```
general-header = Cache-Control            ; Section 14.9
               | Connection               ; Section 14.10
               | Date                     ; Section 14.19
               | Pragma                   ; Section 14.32
               | Transfer-Encoding        ; Section 14.40
               | Upgrade                  ; Section 14.41
               | Via                      ; Section 14.44
```

General-header field names can be extended reliably only in combination with a change in the protocol version. However, new or experimental header fields may be given the semantics of general header fields if all parties in the communication recognize them to be general-header fields. Unrecognized header fields are treated as entity-header fields.

5 Request

A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use.

```
Request       = Request-Line              ; Section 5.1
                *( general-header         ; Section 4.5
                 | request-header         ; Section 5.3
                 | entity-header )        ; Section 7.1
                CRLF
                [ message-body ]          ; Section 7.2
```

5.1 Request-Line

The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by SP characters. No CR or LF are allowed except in the final CRLF sequence.

```
Request-Line   = Method SP Request-URI SP HTTP-Version CRLF
```

5.1.1 Method

The Method token indicates the method to be performed on the resource identified by the Request-URI. The method is case-sensitive.

```
Method         = "OPTIONS"                ; Section 9.2
               | "GET"                    ; Section 9.3
               | "HEAD"                   ; Section 9.4
               | "POST"                   ; Section 9.5
               | "PUT"                    ; Section 9.6
               | "DELETE"                 ; Section 9.7
               | "TRACE"                  ; Section 9.8
               | extension-method

extension-method = token
```

The list of methods allowed by a resource can be specified in an Allow header field (section 14.7). The return code of the response always notifies the client whether a method is currently allowed on a resource, since the set of allowed methods can change dynamically. Servers SHOULD return the status code 405 (Method Not Allowed) if the method is known by the server but not allowed for the requested resource, and 501 (Not Implemented) if the method is unrecognized or not implemented by the server. The list of methods known by a server can be listed in a Public response-header field (section 14.35).

The methods GET and HEAD MUST be supported by all general-purpose servers. All other methods are optional; however, if the above methods are implemented, they MUST be implemented with the same semantics as those specified in section 9.

5.1.2 Request-URI

The Request-URI is a Uniform Resource Identifier (section 3.2) and identifies the resource upon which to apply the request.

```
Request-URI    = "*" | absoluteURI | abs_path
```

The three options for Request-URI are dependent on the nature of the request. The asterisk "*" means that the request does not apply to a particular resource, but to the server itself, and is only allowed when the method used does not necessarily apply to a resource. One example would be

```
OPTIONS * HTTP/1.1
```

The absoluteURI form is required when the request is being made to a proxy. The proxy is requested to forward the request or service it

from a valid cache, and return the response. Note that the proxy MAY forward the request on to another proxy or directly to the server specified by the absoluteURI. In order to avoid request loops, a proxy MUST be able to recognize all of its server names, including any aliases, local variations, and the numeric IP address. An example Request-Line would be:

```
GET http://www.w3.org/pub/WWW/TheProject.html HTTP/1.1
```

To allow for transition to absoluteURIs in all requests in future versions of HTTP, all HTTP/1.1 servers MUST accept the absoluteURI form in requests, even though HTTP/1.1 clients will only generate them in requests to proxies.

The most common form of Request-URI is that used to identify a resource on an origin server or gateway. In this case the absolute path of the URI MUST be transmitted (see section 3.2.1, abs_path) as the Request-URI, and the network location of the URI (net_loc) MUST be transmitted in a Host header field. For example, a client wishing to retrieve the resource above directly from the origin server would create a TCP connection to port 80 of the host "www.w3.org" and send the lines:

```
GET /pub/WWW/TheProject.html HTTP/1.1
Host: www.w3.org
```

followed by the remainder of the Request. Note that the absolute path cannot be empty; if none is present in the original URI, it MUST be given as "/" (the server root).

If a proxy receives a request without any path in the Request-URI and the method specified is capable of supporting the asterisk form of request, then the last proxy on the request chain MUST forward the request with "*" as the final Request-URI. For example, the request

```
OPTIONS http://www.ics.uci.edu:8001 HTTP/1.1
```

would be forwarded by the proxy as

```
OPTIONS * HTTP/1.1
Host: www.ics.uci.edu:8001
```

after connecting to port 8001 of host "www.ics.uci.edu".

The Request-URI is transmitted in the format specified in section 3.2.1. The origin server MUST decode the Request-URI in order to properly interpret the request. Servers SHOULD respond to invalid Request-URIs with an appropriate status code.

In requests that they forward, proxies MUST NOT rewrite the "abs_path" part of a Request-URI in any way except as noted above to replace a null abs_path with "*", no matter what the proxy does in its internal implementation.

Note: The "no rewrite" rule prevents the proxy from changing the meaning of the request when the origin server is improperly using a non-reserved URL character for a reserved purpose. Implementers should be aware that some pre-HTTP/1.1 proxies have been known to rewrite the Request-URI.

5.2 The Resource Identified by a Request

HTTP/1.1 origin servers SHOULD be aware that the exact resource identified by an Internet request is determined by examining both the Request-URI and the Host header field.

An origin server that does not allow resources to differ by the requested host MAY ignore the Host header field value. (But see section 19.5.1 for other requirements on Host support in HTTP/1.1.)

An origin server that does differentiate resources based on the host requested (sometimes referred to as virtual hosts or vanity hostnames) MUST use the following rules for determining the requested resource on an HTTP/1.1 request:

1. If Request-URI is an absoluteURI, the host is part of the Request-URI. Any Host header field value in the request MUST be ignored.

2. If the Request-URI is not an absoluteURI, and the request includes a Host header field, the host is determined by the Host header field value.

3. If the host as determined by rule 1 or 2 is not a valid host on the server, the response MUST be a 400 (Bad Request) error message.

Recipients of an HTTP/1.0 request that lacks a Host header field MAY attempt to use heuristics (e.g., examination of the URI path for something unique to a particular host) in order to determine what exact resource is being requested.

5.3 Request Header Fields

The request-header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers, with semantics

equivalent to the parameters on a programming language method invocation.

```
       request-header = Accept                   ; Section 14.1
                      | Accept-Charset           ; Section 14.2
                      | Accept-Encoding          ; Section 14.3
                      | Accept-Language          ; Section 14.4
                      | Authorization            ; Section 14.8
                      | From                     ; Section 14.22
                      | Host                     ; Section 14.23
                      | If-Modified-Since        ; Section 14.24
                      | If-Match                 ; Section 14.25
                      | If-None-Match            ; Section 14.26
                      | If-Range                 ; Section 14.27
                      | If-Unmodified-Since      ; Section 14.28
                      | Max-Forwards             ; Section 14.31
                      | Proxy-Authorization      ; Section 14.34
                      | Range                    ; Section 14.36
                      | Referer                  ; Section 14.37
                      | User-Agent               ; Section 14.42
```

Request-header field names can be extended reliably only in combination with a change in the protocol version. However, new or experimental header fields MAY be given the semantics of request-header fields if all parties in the communication recognize them to be request-header fields. Unrecognized header fields are treated as entity-header fields.

6 Response

After receiving and interpreting a request message, a server responds with an HTTP response message.

```
    Response      = Status-Line               ; Section 6.1
                    *( general-header         ; Section 4.5
                     | response-header        ; Section 6.2
                     | entity-header )        ; Section 7.1
                    CRLF
                    [ message-body ]          ; Section 7.2
```

6.1 Status-Line

The first line of a Response message is the Status-Line, consisting of the protocol version followed by a numeric status code and its associated textual phrase, with each element separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

```
      Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

6.1.1 Status Code and Reason Phrase

The Status-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request. These codes are fully defined in section 10. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata and the Reason-Phrase is intended for the human user. The client is not required to examine or display the Reason-Phrase.

The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. There are 5 values for the first digit:

- 1xx: Informational - Request received, continuing process
- 2xx: Success - The action was successfully received, understood, and accepted
- 3xx: Redirection - Further action must be taken in order to complete the request
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error - The server failed to fulfill an apparently valid request

The individual values of the numeric status codes defined for HTTP/1.1, and an example set of corresponding Reason-Phrase's, are presented below. The reason phrases listed here are only recommended -- they may be replaced by local equivalents without affecting the protocol.

```
      Status-Code    = "100"   ; Continue
                     | "101"   ; Switching Protocols
                     | "200"   ; OK
                     | "201"   ; Created
                     | "202"   ; Accepted
                     | "203"   ; Non-Authoritative Information
                     | "204"   ; No Content
                     | "205"   ; Reset Content
                     | "206"   ; Partial Content
                     | "300"   ; Multiple Choices
                     | "301"   ; Moved Permanently
                     | "302"   ; Moved Temporarily
```



```
                     | "303"   ; See Other
                     | "304"   ; Not Modified
                     | "305"   ; Use Proxy
                     | "400"   ; Bad Request
                     | "401"   ; Unauthorized
                     | "402"   ; Payment Required
                     | "403"   ; Forbidden
                     | "404"   ; Not Found
                     | "405"   ; Method Not Allowed
                     | "406"   ; Not Acceptable
                     | "407"   ; Proxy Authentication Required
                     | "408"   ; Request Time-out
                     | "409"   ; Conflict
                     | "410"   ; Gone
                     | "411"   ; Length Required
                     | "412"   ; Precondition Failed
                     | "413"   ; Request Entity Too Large
                     | "414"   ; Request-URI Too Large
                     | "415"   ; Unsupported Media Type
                     | "500"   ; Internal Server Error
                     | "501"   ; Not Implemented
                     | "502"   ; Bad Gateway
                     | "503"   ; Service Unavailable
                     | "504"   ; Gateway Time-out
                     | "505"   ; HTTP Version not supported
                     | extension-code

      extension-code = 3DIGIT

      Reason-Phrase  = *<TEXT, excluding CR, LF>
```

HTTP status codes are extensible. HTTP applications are not required to understand the meaning of all registered status codes, though such understanding is obviously desirable. However, applications MUST understand the class of any status code, as indicated by the first digit, and treat any unrecognized response as being equivalent to the x00 status code of that class, with the exception that an unrecognized response MUST NOT be cached. For example, if an unrecognized status code of 431 is received by the client, it can safely assume that there was something wrong with its request and treat the response as if it had received a 400 status code. In such cases, user agents SHOULD present to the user the entity returned with the response, since that entity is likely to include human-readable information which will explain the unusual status.

6.2 Response Header Fields

The response-header fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

```
       response-header = Age                    ; Section 14.6
                       | Location               ; Section 14.30
                       | Proxy-Authenticate     ; Section 14.33
                       | Public                 ; Section 14.35
                       | Retry-After            ; Section 14.38
                       | Server                 ; Section 14.39
                       | Vary                   ; Section 14.43
                       | Warning                ; Section 14.45
                       | WWW-Authenticate       ; Section 14.46
```

Response-header field names can be extended reliably only in combination with a change in the protocol version. However, new or experimental header fields MAY be given the semantics of response-header fields if all parties in the communication recognize them to be response-header fields. Unrecognized header fields are treated as entity-header fields.

7 Entity

Request and Response messages MAY transfer an entity if not otherwise restricted by the request method or response status code. An entity consists of entity-header fields and an entity-body, although some responses will only include the entity-headers.

In this section, both sender and recipient refer to either the client or the server, depending on who sends and who receives the entity.

7.1 Entity Header Fields

Entity-header fields define optional metainformation about the entity-body or, if no body is present, about the resource identified by the request.


```
       entity-header  = Allow                    ; Section 14.7
                      | Content-Base             ; Section 14.11
                      | Content-Encoding         ; Section 14.12
                      | Content-Language         ; Section 14.13
                      | Content-Length           ; Section 14.14
                      | Content-Location         ; Section 14.15
                      | Content-MD5              ; Section 14.16
                      | Content-Range            ; Section 14.17
                      | Content-Type             ; Section 14.18
                      | ETag                     ; Section 14.20
                      | Expires                  ; Section 14.21
                      | Last-Modified            ; Section 14.29
                      | extension-header

       extension-header = message-header
```

The extension-header mechanism allows additional entity-header fields to be defined without changing the protocol, but these fields cannot be assumed to be recognizable by the recipient. Unrecognized header fields SHOULD be ignored by the recipient and forwarded by proxies.

7.2 Entity Body

The entity-body (if any) sent with an HTTP request or response is in a format and encoding defined by the entity-header fields.

```
       entity-body    = *OCTET
```

An entity-body is only present in a message when a message-body is present, as described in section 4.3. The entity-body is obtained from the message-body by decoding any Transfer-Encoding that may have been applied to ensure safe and proper transfer of the message.

7.2.1 Type

When an entity-body is included with a message, the data type of that body is determined via the header fields Content-Type and Content-Encoding. These define a two-layer, ordered encoding model:

```
       entity-body := Content-Encoding( Content-Type( data ) )
```

Content-Type specifies the media type of the underlying data. Content-Encoding may be used to indicate any additional content codings applied to the data, usually for the purpose of data compression, that are a property of the requested resource. There is no default encoding.

Any HTTP/1.1 message containing an entity-body SHOULD include a Content-Type header field defining the media type of that body. If and only if the media type is not given by a Content-Type field, the recipient MAY attempt to guess the media type via inspection of its content and/or the name extension(s) of the URL used to identify the resource. If the media type remains unknown, the recipient SHOULD treat it as type "application/octet-stream".

7.2.2 Length

The length of an entity-body is the length of the message-body after any transfer codings have been removed. Section 4.4 defines how the length of a message-body is determined.

8 Connections

8.1 Persistent Connections

8.1.1 Purpose

Prior to persistent connections, a separate TCP connection was established to fetch each URL, increasing the load on HTTP servers and causing congestion on the Internet. The use of inline images and other associated data often requires a client to make multiple requests of the same server in a short amount of time. Analyses of these performance problems are available [30][27]; analysis and results from a prototype implementation are in [26].

Persistent HTTP connections have a number of advantages:

- By opening and closing fewer TCP connections, CPU time is saved, and memory used for TCP protocol control blocks is also saved.
- HTTP requests and responses can be pipelined on a connection. Pipelining allows a client to make multiple requests without waiting for each response, allowing a single TCP connection to be used much more efficiently, with much lower elapsed time.
- Network congestion is reduced by reducing the number of packets caused by TCP opens, and by allowing TCP sufficient time to determine the congestion state of the network.
- HTTP can evolve more gracefully; since errors can be reported without the penalty of closing the TCP connection. Clients using future versions of HTTP might optimistically try a new feature, but if communicating with an older server, retry with old semantics after an error is reported.

HTTP implementations SHOULD implement persistent connections.

8.1.2 Overall Operation

A significant difference between HTTP/1.1 and earlier versions of HTTP is that persistent connections are the default behavior of any HTTP connection. That is, unless otherwise indicated, the client may assume that the server will maintain a persistent connection.

Persistent connections provide a mechanism by which a client and a server can signal the close of a TCP connection. This signaling takes place using the Connection header field. Once a close has been signaled, the client MUST not send any more requests on that connection.

8.1.2.1 Negotiation

An HTTP/1.1 server MAY assume that a HTTP/1.1 client intends to maintain a persistent connection unless a Connection header including the connection-token "close" was sent in the request. If the server chooses to close the connection immediately after sending the response, it SHOULD send a Connection header including the connection-token close.

An HTTP/1.1 client MAY expect a connection to remain open, but would decide to keep it open based on whether the response from a server contains a Connection header with the connection-token close. In case the client does not want to maintain a connection for more than that request, it SHOULD send a Connection header including the connection-token close.

If either the client or the server sends the close token in the Connection header, that request becomes the last one for the connection.

Clients and servers SHOULD NOT assume that a persistent connection is maintained for HTTP versions less than 1.1 unless it is explicitly signaled. See section 19.7.1 for more information on backwards compatibility with HTTP/1.0 clients.

In order to remain persistent, all messages on the connection must have a self-defined message length (i.e., one not defined by closure of the connection), as described in section 4.4.

8.1.2.2 Pipelining

A client that supports persistent connections MAY "pipeline" its requests (i.e., send multiple requests without waiting for each response). A server MUST send its responses to those requests in the same order that the requests were received.

Clients which assume persistent connections and pipeline immediately after connection establishment SHOULD be prepared to retry their connection if the first pipelined attempt fails. If a client does such a retry, it MUST NOT pipeline before it knows the connection is persistent. Clients MUST also be prepared to resend their requests if the server closes the connection before sending all of the corresponding responses.

8.1.3 Proxy Servers

It is especially important that proxies correctly implement the properties of the Connection header field as specified in 14.2.1.

The proxy server MUST signal persistent connections separately with its clients and the origin servers (or other proxy servers) that it connects to. Each persistent connection applies to only one transport link.

A proxy server MUST NOT establish a persistent connection with an HTTP/1.0 client.

8.1.4 Practical Considerations

Servers will usually have some time-out value beyond which they will no longer maintain an inactive connection. Proxy servers might make this a higher value since it is likely that the client will be making more connections through the same server. The use of persistent connections places no requirements on the length of this time-out for either the client or the server.

When a client or server wishes to time-out it SHOULD issue a graceful close on the transport connection. Clients and servers SHOULD both constantly watch for the other side of the transport close, and respond to it as appropriate. If a client or server does not detect the other side's close promptly it could cause unnecessary resource drain on the network.

A client, server, or proxy MAY close the transport connection at any time. For example, a client MAY have started to send a new request at the same time that the server has decided to close the "idle" connection. From the server's point of view, the connection is being closed while it was idle, but from the client's point of view, a request is in progress.

This means that clients, servers, and proxies MUST be able to recover from asynchronous close events. Client software SHOULD reopen the transport connection and retransmit the aborted request without user interaction so long as the request method is idempotent (see section

9.1.2); other methods MUST NOT be automatically retried, although user agents MAY offer a human operator the choice of retrying the request.

However, this automatic retry SHOULD NOT be repeated if the second request fails.

Servers SHOULD always respond to at least one request per connection, if at all possible. Servers SHOULD NOT close a connection in the middle of transmitting a response, unless a network or client failure is suspected.

Clients that use persistent connections SHOULD limit the number of simultaneous connections that they maintain to a given server. A single-user client SHOULD maintain AT MOST 2 connections with any server or proxy. A proxy SHOULD use up to 2*N connections to another server or proxy, where N is the number of simultaneously active users. These guidelines are intended to improve HTTP response times and avoid congestion of the Internet or other networks.

8.2 Message Transmission Requirements

General requirements:

- HTTP/1.1 servers SHOULD maintain persistent connections and use TCP's flow control mechanisms to resolve temporary overloads, rather than terminating connections with the expectation that clients will retry. The latter technique can exacerbate network congestion.

- An HTTP/1.1 (or later) client sending a message-body SHOULD monitor the network connection for an error status while it is transmitting the request. If the client sees an error status, it SHOULD immediately cease transmitting the body. If the body is being sent using a "chunked" encoding (section 3.6), a zero length chunk and empty footer MAY be used to prematurely mark the end of the message. If the body was preceded by a Content-Length header, the client MUST close the connection.

- An HTTP/1.1 (or later) client MUST be prepared to accept a 100 (Continue) status followed by a regular response.

- An HTTP/1.1 (or later) server that receives a request from a HTTP/1.0 (or earlier) client MUST NOT transmit the 100 (continue) response; it SHOULD either wait for the request to be completed normally (thus avoiding an interrupted request) or close the connection prematurely.

Upon receiving a method subject to these requirements from an HTTP/1.1 (or later) client, an HTTP/1.1 (or later) server MUST either respond with 100 (Continue) status and continue to read from the input stream, or respond with an error status. If it responds with an error status, it MAY close the transport (TCP) connection or it MAY continue to read and discard the rest of the request. It MUST NOT perform the requested method if it returns an error status.

Clients SHOULD remember the version number of at least the most recently used server; if an HTTP/1.1 client has seen an HTTP/1.1 or later response from the server, and it sees the connection close before receiving any status from the server, the client SHOULD retry the request without user interaction so long as the request method is idempotent (see section 9.1.2); other methods MUST NOT be automatically retried, although user agents MAY offer a human operator the choice of retrying the request.. If the client does retry the request, the client

- MUST first send the request header fields, and then
- MUST wait for the server to respond with either a 100 (Continue) response, in which case the client should continue, or with an error status.

If an HTTP/1.1 client has not seen an HTTP/1.1 or later response from the server, it should assume that the server implements HTTP/1.0 or older and will not use the 100 (Continue) response. If in this case the client sees the connection close before receiving any status from the server, the client SHOULD retry the request. If the client does retry the request to this HTTP/1.0 server, it should use the following "binary exponential backoff" algorithm to be assured of obtaining a reliable response:

1. Initiate a new connection to the server
2. Transmit the request-headers
3. Initialize a variable R to the estimated round-trip time to the server (e.g., based on the time it took to establish the connection), or to a constant value of 5 seconds if the round-trip time is not available.
4. Compute T = R * (2**N), where N is the number of previous retries of this request.
5. Wait either for an error response from the server, or for T seconds (whichever comes first)

6. If no error response is received, after T seconds transmit the body of the request.
7. If client sees that the connection is closed prematurely, repeat from step 1 until the request is accepted, an error response is received, or the user becomes impatient and terminates the retry process.

No matter what the server version, if an error status is received, the client

- MUST NOT continue and
- MUST close the connection if it has not completed sending the message.

An HTTP/1.1 (or later) client that sees the connection close after receiving a 100 (Continue) but before receiving any other status SHOULD retry the request, and need not wait for 100 (Continue) response (but MAY do so if this simplifies the implementation).

9 Method Definitions

The set of common methods for HTTP/1.1 is defined below. Although this set can be expanded, additional methods cannot be assumed to share the same semantics for separately extended clients and servers.

The Host request-header field (section 14.23) MUST accompany all HTTP/1.1 requests.

9.1 Safe and Idempotent Methods

9.1.1 Safe Methods

Implementers should be aware that the software represents the user in their interactions over the Internet, and should be careful to allow the user to be aware of any actions they may take which may have an unexpected significance to themselves or others.

In particular, the convention has been established that the GET and HEAD methods should never have the significance of taking an action other than retrieval. These methods should be considered "safe." This allows user agents to represent other methods, such as POST, PUT and DELETE, in a special way, so that the user is made aware of the fact that a possibly unsafe action is being requested.

Naturally, it is not possible to ensure that the server does not generate side-effects as a result of performing a GET request; in

fact, some dynamic resources consider that a feature. The important distinction here is that the user did not request the side-effects, so therefore cannot be held accountable for them.

9.1.2 Idempotent Methods

Methods may also have the property of "idempotence" in that (aside from error or expiration issues) the side-effects of N > 0 identical requests is the same as for a single request. The methods GET, HEAD, PUT and DELETE share this property.

9.2 OPTIONS

The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.

Unless the server's response is an error, the response MUST NOT include entity information other than what can be considered as communication options (e.g., Allow is appropriate, but Content-Type is not). Responses to this method are not cachable.

If the Request-URI is an asterisk ("*"), the OPTIONS request is intended to apply to the server as a whole. A 200 response SHOULD include any header fields which indicate optional features implemented by the server (e.g., Public), including any extensions not defined by this specification, in addition to any applicable general or response-header fields. As described in section 5.1.2, an "OPTIONS *" request can be applied through a proxy by specifying the destination server in the Request-URI without any path information.

If the Request-URI is not an asterisk, the OPTIONS request applies only to the options that are available when communicating with that resource. A 200 response SHOULD include any header fields which indicate optional features implemented by the server and applicable to that resource (e.g., Allow), including any extensions not defined by this specification, in addition to any applicable general or response-header fields. If the OPTIONS request passes through a proxy, the proxy MUST edit the response to exclude those options which apply to a proxy's capabilities and which are known to be unavailable through that proxy.

9.3 GET

The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

The semantics of the GET method change to a "conditional GET" if the request message includes an If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field. A conditional GET method requests that the entity be transferred only under the circumstances described by the conditional header field(s). The conditional GET method is intended to reduce unnecessary network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring data already held by the client.

The semantics of the GET method change to a "partial GET" if the request message includes a Range header field. A partial GET requests that only part of the entity be transferred, as described in section 14.36. The partial GET method is intended to reduce unnecessary network usage by allowing partially-retrieved entities to be completed without transferring data already held by the client.

The response to a GET request is cachable if and only if it meets the requirements for HTTP caching described in section 13.

9.4 HEAD

The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The metainformation contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining metainformation about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

The response to a HEAD request may be cachable in the sense that the information contained in the response may be used to update a previously cached entity from that resource. If the new field values indicate that the cached entity differs from the current entity (as would be indicated by a change in Content-Length, Content-MD5, ETag or Last-Modified), then the cache MUST treat the cache entry as stale.

9.5 POST

The POST method is used to request that the destination server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:

- Annotation of existing resources;
- Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
- Providing a block of data, such as the result of submitting a form, to a data-handling process;
- Extending a database through an append operation.

The actual function performed by the POST method is determined by the server and is usually dependent on the Request-URI. The posted entity is subordinate to that URI in the same way that a file is subordinate to a directory containing it, a news article is subordinate to a newsgroup to which it is posted, or a record is subordinate to a database.

The action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either 200 (OK) or 204 (No Content) is the appropriate response status, depending on whether or not the response includes an entity that describes the result.

If a resource has been created on the origin server, the response SHOULD be 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a Location header (see section 14.30).

Responses to this method are not cachable, unless the response includes appropriate Cache-Control or Expires header fields. However, the 303 (See Other) response can be used to direct the user agent to retrieve a cachable resource.

POST requests must obey the message transmission requirements set out in section 8.2.

9.6 PUT

The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI. If a new resource is created, the origin server MUST inform the user agent via the 201 (Created) response. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request. If the resource could not be created or modified with the Request-URI, an appropriate error response SHOULD be given that reflects the nature of the problem. The recipient of the entity MUST NOT ignore any Content-* (e.g. Content-Range) headers that it does not understand or implement and MUST return a 501 (Not Implemented) response in such cases.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries should be treated as stale. Responses to this method are not cachable.

The fundamental difference between the POST and PUT requests is reflected in the different meaning of the Request-URI. The URI in a POST request identifies the resource that will handle the enclosed entity. That resource may be a data-accepting process, a gateway to some other protocol, or a separate entity that accepts annotations. In contrast, the URI in a PUT request identifies the entity enclosed with the request -- the user agent knows what URI is intended and the server MUST NOT attempt to apply the request to some other resource. If the server desires that the request be applied to a different URI, it MUST send a 301 (Moved Permanently) response; the user agent MAY then make its own decision regarding whether or not to redirect the request.

A single resource MAY be identified by many different URIs. For example, an article may have a URI for identifying "the current version" which is separate from the URI identifying each particular version. In this case, a PUT request on a general URI may result in several other URIs being defined by the origin server.

HTTP/1.1 does not define how a PUT method affects the state of an origin server.

PUT requests must obey the message transmission requirements set out in section 8.2.

9.7 DELETE

The DELETE method requests that the origin server delete the resource identified by the Request-URI. This method MAY be overridden by human intervention (or other means) on the origin server. The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully. However, the server SHOULD not indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location.

A successful response SHOULD be 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has not yet been enacted, or 204 (No Content) if the response is OK but does not include an entity.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries should be treated as stale. Responses to this method are not cachable.

9.8 TRACE

The TRACE method is used to invoke a remote, application-layer loop-back of the request message. The final recipient of the request SHOULD reflect the message received back to the client as the entity-body of a 200 (OK) response. The final recipient is either the origin server or the first proxy or gateway to receive a Max-Forwards value of zero (0) in the request (see section 14.31). A TRACE request MUST NOT include an entity.

TRACE allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information. The value of the Via header field (section 14.44) is of particular interest, since it acts as a trace of the request chain. Use of the Max-Forwards header field allows the client to limit the length of the request chain, which is useful for testing a chain of proxies forwarding messages in an infinite loop.

If successful, the response SHOULD contain the entire request message in the entity-body, with a Content-Type of "message/http". Responses to this method MUST NOT be cached.

10 Status Code Definitions

Each Status-Code is described below, including a description of which method(s) it can follow and any metainformation required in the response.

10.1 Informational 1xx

This class of status code indicates a provisional response, consisting only of the Status-Line and optional headers, and is terminated by an empty line. Since HTTP/1.0 did not define any 1xx status codes, servers MUST NOT send a 1xx response to an HTTP/1.0 client except under experimental conditions.

10.1.1 100 Continue

The client may continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server. The client SHOULD continue by sending the remainder of the request or, if the request has already been completed, ignore this response. The server MUST send a final response after the request has been completed.

10.1.2 101 Switching Protocols

The server understands and is willing to comply with the client's request, via the Upgrade message header field (section 14.41), for a change in the application protocol being used on this connection. The server will switch protocols to those defined by the response's Upgrade header field immediately after the empty line which terminates the 101 response.

The protocol should only be switched when it is advantageous to do so. For example, switching to a newer version of HTTP is advantageous over older versions, and switching to a real-time, synchronous protocol may be advantageous when delivering resources that use such features.

10.2 Successful 2xx

This class of status code indicates that the client's request was successfully received, understood, and accepted.

10.2.1 200 OK

The request has succeeded. The information returned with the response is dependent on the method used in the request, for example:

GET an entity corresponding to the requested resource is sent in the response;

HEAD the entity-header fields corresponding to the requested resource are sent in the response without any message-body;

POST an entity describing or containing the result of the action;

TRACE an entity containing the request message as received by the end server.

10.2.2 201 Created

The request has been fulfilled and resulted in a new resource being created. The newly created resource can be referenced by the URI(s) returned in the entity of the response, with the most specific URL for the resource given by a Location header field. The origin server MUST create the resource before returning the 201 status code. If the action cannot be carried out immediately, the server should respond with 202 (Accepted) response instead.

10.2.3 202 Accepted

The request has been accepted for processing, but the processing has not been completed. The request MAY or MAY NOT eventually be acted upon, as it MAY be disallowed when processing actually takes place. There is no facility for re-sending a status code from an asynchronous operation such as this.

The 202 response is intentionally non-committal. Its purpose is to allow a server to accept a request for some other process (perhaps a batch-oriented process that is only run once per day) without requiring that the user agent's connection to the server persist until the process is completed. The entity returned with this response SHOULD include an indication of the request's current status and either a pointer to a status monitor or some estimate of when the user can expect the request to be fulfilled.

10.2.4 203 Non-Authoritative Information

The returned metainformation in the entity-header is not the definitive set as available from the origin server, but is gathered from a local or a third-party copy. The set presented MAY be a subset or superset of the original version. For example, including local annotation information about the resource MAY result in a superset of the metainformation known by the origin server. Use of this response code is not required and is only appropriate when the response would otherwise be 200 (OK).

10.2.5 204 No Content

The server has fulfilled the request but there is no new information to send back. If the client is a user agent, it SHOULD NOT change its document view from that which caused the request to be sent. This

response is primarily intended to allow input for actions to take place without causing a change to the user agent's active document view. The response MAY include new metainformation in the form of entity-headers, which SHOULD apply to the document currently in the user agent's active view.

The 204 response MUST NOT include a message-body, and thus is always terminated by the first empty line after the header fields.

10.2.6 205 Reset Content

The server has fulfilled the request and the user agent SHOULD reset the document view which caused the request to be sent. This response is primarily intended to allow input for actions to take place via user input, followed by a clearing of the form in which the input is given so that the user can easily initiate another input action. The response MUST NOT include an entity.

10.2.7 206 Partial Content

The server has fulfilled the partial GET request for the resource. The request must have included a Range header field (section 14.36) indicating the desired range. The response MUST include either a Content-Range header field (section 14.17) indicating the range included with this response, or a multipart/byteranges Content-Type including Content-Range fields for each part. If multipart/byteranges is not used, the Content-Length header field in the response MUST match the actual number of OCTETs transmitted in the message-body.

A cache that does not support the Range and Content-Range headers MUST NOT cache 206 (Partial) responses.

10.3 Redirection 3xx

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. The action required MAY be carried out by the user agent without interaction with the user if and only if the method used in the second request is GET or HEAD. A user agent SHOULD NOT automatically redirect a request more than 5 times, since such redirections usually indicate an infinite loop.

10.3.1 300 Multiple Choices

The requested resource corresponds to any one of a set of representations, each with its own specific location, and agent-driven negotiation information (section 12) is being provided so that the user (or user agent) can select a preferred representation and redirect its request to that location.

Unless it was a HEAD request, the response SHOULD include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice may be performed automatically. However, this specification does not define any standard for such automatic selection.

If the server has a preferred choice of representation, it SHOULD include the specific URL for that representation in the Location field; user agents MAY use the Location field value for automatic redirection. This response is cachable unless indicated otherwise.

10.3.2 301 Moved Permanently

The requested resource has been assigned a new permanent URI and any future references to this resource SHOULD be done using one of the returned URIs. Clients with link editing capabilities SHOULD automatically re-link references to the Request-URI to one or more of the new references returned by the server, where possible. This response is cachable unless indicated otherwise.

If the new URI is a location, its URL SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

If the 301 status code is received in response to a request other than GET or HEAD, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

> Note: When automatically redirecting a POST request after receiving a 301 status code, some existing HTTP/1.0 user agents will erroneously change it into a GET request.

10.3.3 302 Moved Temporarily

The requested resource resides temporarily under a different URI. Since the redirection may be altered on occasion, the client SHOULD continue to use the Request-URI for future requests. This response is only cachable if indicated by a Cache-Control or Expires header field.

If the new URI is a location, its URL SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

If the 302 status code is received in response to a request other than GET or HEAD, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

> Note: When automatically redirecting a POST request after receiving a 302 status code, some existing HTTP/1.0 user agents will erroneously change it into a GET request.

10.3.4 303 See Other

The response to the request can be found under a different URI and SHOULD be retrieved using a GET method on that resource. This method exists primarily to allow the output of a POST-activated script to redirect the user agent to a selected resource. The new URI is not a substitute reference for the originally requested resource. The 303 response is not cachable, but the response to the second (redirected) request MAY be cachable.

If the new URI is a location, its URL SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

10.3.5 304 Not Modified

If the client has performed a conditional GET request and access is allowed, but the document has not been modified, the server SHOULD respond with this status code. The response MUST NOT contain a message-body.

The response MUST include the following header fields:

- Date
- ETag and/or Content-Location, if the header would have been sent in a 200 response to the same request
- Expires, Cache-Control, and/or Vary, if the field-value might differ from that sent in any previous response for the same variant

If the conditional GET used a strong cache validator (see section 13.3.3), the response SHOULD NOT include other entity-headers. Otherwise (i.e., the conditional GET used a weak validator), the response MUST NOT include other entity-headers; this prevents inconsistencies between cached entity-bodies and updated headers.

If a 304 response indicates an entity not currently cached, then the cache MUST disregard the response and repeat the request without the conditional.

If a cache uses a received 304 response to update a cache entry, the cache MUST update the entry to reflect any new field values given in the response.

The 304 response MUST NOT include a message-body, and thus is always terminated by the first empty line after the header fields.

10.3.6 305 Use Proxy

The requested resource MUST be accessed through the proxy given by the Location field. The Location field gives the URL of the proxy. The recipient is expected to repeat the request via the proxy.

10.4 Client Error 4xx

The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method. User agents SHOULD display any included entity to the user.

> Note: If the client is sending data, a server implementation using TCP should be careful to ensure that the client acknowledges receipt of the packet(s) containing the response, before the server closes the input connection. If the client continues sending data to the server after the close, the server's TCP stack will send a reset packet to the client, which may erase the client's unacknowledged input buffers before they can be read and interpreted by the HTTP application.

10.4.1 400 Bad Request

The request could not be understood by the server due to malformed syntax. The client SHOULD NOT repeat the request without modifications.

10.4.2 401 Unauthorized

The request requires user authentication. The response MUST include a WWW-Authenticate header field (section 14.46) containing a challenge applicable to the requested resource. The client MAY repeat the request with a suitable Authorization header field (section 14.8). If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials. If the 401 response contains the same challenge as the prior response, and the user agent has already attempted authentication at least once, then the user SHOULD be presented the entity that was given in the response, since that entity MAY include relevant diagnostic information. HTTP access authentication is explained in section 11.

10.4.3 402 Payment Required

This code is reserved for future use.

10.4.4 403 Forbidden

The server understood the request, but is refusing to fulfill it. Authorization will not help and the request SHOULD NOT be repeated. If the request method was not HEAD and the server wishes to make public why the request has not been fulfilled, it SHOULD describe the reason for the refusal in the entity. This status code is commonly used when the server does not wish to reveal exactly why the request has been refused, or when no other response is applicable.

10.4.5 404 Not Found

The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent.

If the server does not wish to make this information available to the client, the status code 403 (Forbidden) can be used instead. The 410 (Gone) status code SHOULD be used if the server knows, through some internally configurable mechanism, that an old resource is permanently unavailable and has no forwarding address.

10.4.6 405 Method Not Allowed

The method specified in the Request-Line is not allowed for the resource identified by the Request-URI. The response MUST include an Allow header containing a list of valid methods for the requested resource.

10.4.7 406 Not Acceptable

The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.

Unless it was a HEAD request, the response SHOULD include an entity containing a list of available entity characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice may be performed automatically. However, this specification does not define any standard for such automatic selection.

> Note: HTTP/1.1 servers are allowed to return responses which are not acceptable according to the accept headers sent in the request. In some cases, this may even be preferable to sending a 406 response. User agents are encouraged to inspect the headers of an incoming response to determine if it is acceptable. If the response could be unacceptable, a user agent SHOULD temporarily stop receipt of more data and query the user for a decision on further actions.

10.4.8 407 Proxy Authentication Required

This code is similar to 401 (Unauthorized), but indicates that the client MUST first authenticate itself with the proxy. The proxy MUST return a Proxy-Authenticate header field (section 14.33) containing a challenge applicable to the proxy for the requested resource. The client MAY repeat the request with a suitable Proxy-Authorization header field (section 14.34). HTTP access authentication is explained in section 11.

10.4.9 408 Request Timeout

The client did not produce a request within the time that the server was prepared to wait. The client MAY repeat the request without modifications at any later time.

10.4.10 409 Conflict

The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations where it is expected that the user might be able to resolve the conflict and resubmit the request. The response body SHOULD include enough information for the user to recognize the source of the conflict. Ideally, the response entity would include enough information for the user or user agent to fix the problem; however, that may not be possible and is not required.

Conflicts are most likely to occur in response to a PUT request. If versioning is being used and the entity being PUT includes changes to a resource which conflict with those made by an earlier (third-party) request, the server MAY use the 409 response to indicate that it can't complete the request. In this case, the response entity SHOULD contain a list of the differences between the two versions in a format defined by the response Content-Type.

10.4.11 410 Gone

The requested resource is no longer available at the server and no forwarding address is known. This condition SHOULD be considered permanent. Clients with link editing capabilities SHOULD delete references to the Request-URI after user approval. If the server does not know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (Not Found) SHOULD be used instead. This response is cachable unless indicated otherwise.

The 410 response is primarily intended to assist the task of web maintenance by notifying the recipient that the resource is intentionally unavailable and that the server owners desire that remote links to that resource be removed. Such an event is common for limited-time, promotional services and for resources belonging to individuals no longer working at the server's site. It is not necessary to mark all permanently unavailable resources as "gone" or to keep the mark for any length of time -- that is left to the discretion of the server owner.

10.4.12 411 Length Required

The server refuses to accept the request without a defined Content-Length. The client MAY repeat the request if it adds a valid Content-Length header field containing the length of the message-body in the request message.

10.4.13 412 Precondition Failed

The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server. This response code allows the client to place preconditions on the current resource metainformation (header field data) and thus prevent the requested method from being applied to a resource other than the one intended.

10.4.14 413 Request Entity Too Large

The server is refusing to process a request because the request entity is larger than the server is willing or able to process. The server may close the connection to prevent the client from continuing the request.

If the condition is temporary, the server SHOULD include a Retry-After header field to indicate that it is temporary and after what time the client may try again.

10.4.15 414 Request-URI Too Long

The server is refusing to service the request because the Request-URI is longer than the server is willing to interpret. This rare condition is only likely to occur when a client has improperly converted a POST request to a GET request with long query information, when the client has descended into a URL "black hole" of redirection (e.g., a redirected URL prefix that points to a suffix of itself), or when the server is under attack by a client attempting to exploit security holes present in some servers using fixed-length buffers for reading or manipulating the Request-URI.

10.4.16 415 Unsupported Media Type

The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.

10.5 Server Error 5xx

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents SHOULD display any included entity to the user. These response codes are applicable to any request method.

10.5.1 500 Internal Server Error

The server encountered an unexpected condition which prevented it from fulfilling the request.

10.5.2 501 Not Implemented

The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource.

10.5.3 502 Bad Gateway

The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.

10.5.4 503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay may be indicated in a Retry-After header. If no Retry-After is given, the client SHOULD handle the response as it would for a 500 response.

Note: The existence of the 503 status code does not imply that a server must use it when becoming overloaded. Some servers may wish to simply refuse the connection.

10.5.5 504 Gateway Timeout

The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server it accessed in attempting to complete the request.

10.5.6 505 HTTP Version Not Supported

The server does not support, or refuses to support, the HTTP protocol version that was used in the request message. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client, as described in section 3.1, other than with this error message. The response SHOULD contain an entity describing why that version is not supported and what other protocols are supported by that server.

11 Access Authentication

HTTP provides a simple challenge-response authentication mechanism which MAY be used by a server to challenge a client request and by a client to provide authentication information. It uses an extensible, case-insensitive token to identify the authentication scheme, followed by a comma-separated list of attribute-value pairs which carry the parameters necessary for achieving authentication via that scheme.

```
auth-scheme    = token

auth-param     = token "=" quoted-string
```

The 401 (Unauthorized) response message is used by an origin server to challenge the authorization of a user agent. This response MUST include a WWW-Authenticate header field containing at least one challenge applicable to the requested resource.

```
challenge      = auth-scheme 1*SP realm *( "," auth-param )

realm          = "realm" "=" realm-value
realm-value    = quoted-string
```

The realm attribute (case-insensitive) is required for all authentication schemes which issue a challenge. The realm value (case-sensitive), in combination with the canonical root URL (see section 5.1.2) of the server being accessed, defines the protection space. These realms allow the protected resources on a server to be partitioned into a set of protection spaces, each with its own authentication scheme and/or authorization database. The realm value is a string, generally assigned by the origin server, which may have additional semantics specific to the authentication scheme.

A user agent that wishes to authenticate itself with a server--usually, but not necessarily, after receiving a 401 or 411 response--MAY do so by including an Authorization header field with the request. The Authorization field value consists of credentials

containing the authentication information of the user agent for the realm of the resource being requested.

```
credentials    = basic-credentials
               | auth-scheme #auth-param
```

The domain over which credentials can be automatically applied by a user agent is determined by the protection space. If a prior request has been authorized, the same credentials MAY be reused for all other requests within that protection space for a period of time determined by the authentication scheme, parameters, and/or user preference. Unless otherwise defined by the authentication scheme, a single protection space cannot extend outside the scope of its server.

If the server does not wish to accept the credentials sent with a request, it SHOULD return a 401 (Unauthorized) response. The response MUST include a WWW-Authenticate header field containing the (possibly new) challenge applicable to the requested resource and an entity explaining the refusal.

The HTTP protocol does not restrict applications to this simple challenge-response mechanism for access authentication. Additional mechanisms MAY be used, such as encryption at the transport level or via message encapsulation, and with additional header fields specifying authentication information. However, these additional mechanisms are not defined by this specification.

Proxies MUST be completely transparent regarding user agent authentication. That is, they MUST forward the WWW-Authenticate and Authorization headers untouched, and follow the rules found in section 14.8.

HTTP/1.1 allows a client to pass authentication information to and from a proxy via the Proxy-Authenticate and Proxy-Authorization headers.

11.1 Basic Authentication Scheme

The "basic" authentication scheme is based on the model that the user agent must authenticate itself with a user-ID and a password for each realm. The realm value should be considered an opaque string which can only be compared for equality with other realms on that server. The server will service the request only if it can validate the user-ID and password for the protection space of the Request-URI. There are no optional authentication parameters.

Upon receipt of an unauthorized request for a URI within the protection space, the server MAY respond with a challenge like the following:

```
WWW-Authenticate: Basic realm="WallyWorld"
```

where "WallyWorld" is the string assigned by the server to identify the protection space of the Request-URI.

To receive authorization, the client sends the userid and password, separated by a single colon (":") character, within a base64 encoded string in the credentials.

```
basic-credentials = "Basic" SP basic-cookie

basic-cookie   = <base64 [7] encoding of user-pass,
                 except not limited to 76 char/line>

user-pass   = userid ":" password

userid      = *<TEXT excluding ":">

password    = *TEXT
```

Userids might be case sensitive.

If the user agent wishes to send the userid "Aladdin" and password "open sesame", it would use the following header field:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

See section 15 for security considerations associated with Basic authentication.

11.2 Digest Authentication Scheme

A digest authentication for HTTP is specified in RFC 2069 [32].

12 Content Negotiation

Most HTTP responses include an entity which contains information for interpretation by a human user. Naturally, it is desirable to supply the user with the "best available" entity corresponding to the request. Unfortunately for servers and caches, not all users have the same preferences for what is "best," and not all user agents are equally capable of rendering all entity types. For that reason, HTTP has provisions for several mechanisms for "content negotiation" -- the process of selecting the best representation for a given response

when there are multiple representations available.

> Note: This is not called "format negotiation" because the alternate representations may be of the same media type, but use different capabilities of that type, be in different languages, etc.

Any response containing an entity-body MAY be subject to negotiation, including error responses.

There are two kinds of content negotiation which are possible in HTTP: server-driven and agent-driven negotiation. These two kinds of negotiation are orthogonal and thus may be used separately or in combination. One method of combination, referred to as transparent negotiation, occurs when a cache uses the agent-driven negotiation information provided by the origin server in order to provide server-driven negotiation for subsequent requests.

12.1 Server-driven Negotiation

If the selection of the best representation for a response is made by an algorithm located at the server, it is called server-driven negotiation. Selection is based on the available representations of the response (the dimensions over which it can vary; e.g. language, content-coding, etc.) and the contents of particular header fields in the request message or on other information pertaining to the request (such as the network address of the client).

Server-driven negotiation is advantageous when the algorithm for selecting from among the available representations is difficult to describe to the user agent, or when the server desires to send its "best guess" to the client along with the first response (hoping to avoid the round-trip delay of a subsequent request if the "best guess" is good enough for the user). In order to improve the server's guess, the user agent MAY include request header fields (Accept, Accept-Language, Accept-Encoding, etc.) which describe its preferences for such a response.

Server-driven negotiation has disadvantages:

1. It is impossible for the server to accurately determine what might be "best" for any given user, since that would require complete knowledge of both the capabilities of the user agent and the intended use for the response (e.g., does the user want to view it on screen or print it on paper?).

2. Having the user agent describe its capabilities in every request can be both very inefficient (given that only a small percentage of responses have multiple representations) and a potential violation of

the user's privacy.

3. It complicates the implementation of an origin server and the algorithms for generating responses to a request.

4. It may limit a public cache's ability to use the same response for multiple user's requests.

HTTP/1.1 includes the following request-header fields for enabling server-driven negotiation through description of user agent capabilities and user preferences: Accept (section 14.1), Accept-Charset (section 14.2), Accept-Encoding (section 14.3), Accept-Language (section 14.4), and User-Agent (section 14.42). However, an origin server is not limited to these dimensions and MAY vary the response based on any aspect of the request, including information outside the request-header fields or within extension header fields not defined by this specification.

HTTP/1.1 origin servers MUST include an appropriate Vary header field (section 14.43) in any cachable response based on server-driven negotiation. The Vary header field describes the dimensions over which the response might vary (i.e. the dimensions over which the origin server picks its "best guess" response from multiple representations).

HTTP/1.1 public caches MUST recognize the Vary header field when it is included in a response and obey the requirements described in section 13.6 that describes the interactions between caching and content negotiation.

12.2 Agent-driven Negotiation

With agent-driven negotiation, selection of the best representation for a response is performed by the user agent after receiving an initial response from the origin server. Selection is based on a list of the available representations of the response included within the header fields (this specification reserves the field-name Alternates, as described in appendix 19.6.2.1) or entity-body of the initial response, with each representation identified by its own URI. Selection from among the representations may be performed automatically (if the user agent is capable of doing so) or manually by the user selecting from a generated (possibly hypertext) menu.

Agent-driven negotiation is advantageous when the response would vary over commonly-used dimensions (such as type, language, or encoding), when the origin server is unable to determine a user agent's capabilities from examining the request, and generally when public caches are used to distribute server load and reduce network usage.

Agent-driven negotiation suffers from the disadvantage of needing a second request to obtain the best alternate representation. This second request is only efficient when caching is used. In addition, this specification does not define any mechanism for supporting automatic selection, though it also does not prevent any such mechanism from being developed as an extension and used within HTTP/1.1.

HTTP/1.1 defines the 300 (Multiple Choices) and 406 (Not Acceptable) status codes for enabling agent-driven negotiation when the server is unwilling or unable to provide a varying response using server-driven negotiation.

12.3 Transparent Negotiation

Transparent negotiation is a combination of both server-driven and agent-driven negotiation. When a cache is supplied with a form of the list of available representations of the response (as in agent-driven negotiation) and the dimensions of variance are completely understood by the cache, then the cache becomes capable of performing server-driven negotiation on behalf of the origin server for subsequent requests on that resource.

Transparent negotiation has the advantage of distributing the negotiation work that would otherwise be required of the origin server and also removing the second request delay of agent-driven negotiation when the cache is able to correctly guess the right response.

This specification does not define any mechanism for transparent negotiation, though it also does not prevent any such mechanism from being developed as an extension and used within HTTP/1.1. An HTTP/1.1 cache performing transparent negotiation MUST include a Vary header field in the response (defining the dimensions of its variance) if it is cachable to ensure correct interoperation with all HTTP/1.1 clients. The agent-driven negotiation information supplied by the origin server SHOULD be included with the transparently negotiated response.

13 Caching in HTTP

HTTP is typically used for distributed information systems, where performance can be improved by the use of response caches. The HTTP/1.1 protocol includes a number of elements intended to make caching work as well as possible. Because these elements are inextricable from other aspects of the protocol, and because they interact with each other, it is useful to describe the basic caching design of HTTP separately from the detailed descriptions of methods,

headers, response codes, etc.

Caching would be useless if it did not significantly improve performance. The goal of caching in HTTP/1.1 is to eliminate the need to send requests in many cases, and to eliminate the need to send full responses in many other cases. The former reduces the number of network round-trips required for many operations; we use an "expiration" mechanism for this purpose (see section 13.2). The latter reduces network bandwidth requirements; we use a "validation" mechanism for this purpose (see section 13.3).

Requirements for performance, availability, and disconnected operation require us to be able to relax the goal of semantic transparency. The HTTP/1.1 protocol allows origin servers, caches, and clients to explicitly reduce transparency when necessary. However, because non-transparent operation may confuse non-expert users, and may be incompatible with certain server applications (such as those for ordering merchandise), the protocol requires that transparency be relaxed

- only by an explicit protocol-level request when relaxed by client or origin server

- only with an explicit warning to the end user when relaxed by cache or client

Therefore, the HTTP/1.1 protocol provides these important elements:

1. Protocol features that provide full semantic transparency when this is required by all parties.

2. Protocol features that allow an origin server or user agent to explicitly request and control non-transparent operation.

3. Protocol features that allow a cache to attach warnings to responses that do not preserve the requested approximation of semantic transparency.

A basic principle is that it must be possible for the clients to detect any potential relaxation of semantic transparency.

Note: The server, cache, or client implementer may be faced with design decisions not explicitly discussed in this specification. If a decision may affect semantic transparency, the implementer ought to err on the side of maintaining transparency unless a careful and complete analysis shows significant benefits in breaking transparency.

13.1.1 Cache Correctness

A correct cache MUST respond to a request with the most up-to-date response held by the cache that is appropriate to the request (see sections 13.2.5, 13.2.6, and 13.12) which meets one of the following conditions:

1. It has been checked for equivalence with what the origin server would have returned by revalidating the response with the origin server (section 13.3);

2. It is "fresh enough" (see section 13.2). In the default case, this means it meets the least restrictive freshness requirement of the client, server, and cache (see section 14.9); if the origin server so specifies, it is the freshness requirement of the origin server alone.

3. It includes a warning if the freshness demand of the client or the origin server is violated (see section 13.1.5 and 14.45).

4. It is an appropriate 304 (Not Modified), 305 (Proxy Redirect), or error (4xx or 5xx) response message.

If the cache can not communicate with the origin server, then a correct cache SHOULD respond as above if the response can be correctly served from the cache; if not it MUST return an error or

warning indicating that there was a communication failure.

If a cache receives a response (either an entire response, or a 304 (Not Modified) response) that it would normally forward to the requesting client, and the received response is no longer fresh, the cache SHOULD forward it to the requesting client without adding a new Warning (but without removing any existing Warning headers). A cache SHOULD NOT attempt to revalidate a response simply because that response became stale in transit; this might lead to an infinite loop. An user agent that receives a stale response without a Warning MAY display a warning indication to the user.

13.1.2 Warnings

Whenever a cache returns a response that is neither first-hand nor "fresh enough" (in the sense of condition 2 in section 13.1.1), it must attach a warning to that effect, using a Warning response-header. This warning allows clients to take appropriate action.

Warnings may be used for other purposes, both cache-related and otherwise. The use of a warning, rather than an error status code, distinguish these responses from true failures.

Warnings are always cachable, because they never weaken the transparency of a response. This means that warnings can be passed to HTTP/1.0 caches without danger; such caches will simply pass the warning along as an entity-header in the response.

Warnings are assigned numbers between 0 and 99. This specification defines the code numbers and meanings of each currently assigned warnings, allowing a client or cache to take automated action in some (but not all) cases.

Warnings also carry a warning text. The text may be in any appropriate natural language (perhaps based on the client's Accept headers), and include an optional indication of what character set is used.

Multiple warnings may be attached to a response (either by the origin server or by a cache), including multiple warnings with the same code number. For example, a server may provide the same warning with texts in both English and Basque.

When multiple warnings are attached to a response, it may not be practical or reasonable to display all of them to the user. This version of HTTP does not specify strict priority rules for deciding which warnings to display and in what order, but does suggest some heuristics.

The Warning header and the currently defined warnings are described in section 14.45.

13.1.3 Cache-control Mechanisms

The basic cache mechanisms in HTTP/1.1 (server-specified expiration times and validators) are implicit directives to caches. In some cases, a server or client may need to provide explicit directives to the HTTP caches. We use the Cache-Control header for this purpose.

The Cache-Control header allows a client or server to transmit a variety of directives in either requests or responses. These directives typically override the default caching algorithms. As a general rule, if there is any apparent conflict between header values, the most restrictive interpretation should be applied (that is, the one that is most likely to preserve semantic transparency). However, in some cases, Cache-Control directives are explicitly specified as weakening the approximation of semantic transparency (for example, "max-stale" or "public").

The Cache-Control directives are described in detail in section 14.9.

13.1.4 Explicit User Agent Warnings

Many user agents make it possible for users to override the basic caching mechanisms. For example, the user agent may allow the user to specify that cached entities (even explicitly stale ones) are never validated. Or the user agent might habitually add "Cache-Control: max-stale=3600" to every request. The user should have to explicitly request either non-transparent behavior, or behavior that results in abnormally ineffective caching.

If the user has overridden the basic caching mechanisms, the user agent should explicitly indicate to the user whenever this results in the display of information that might not meet the server's transparency requirements (in particular, if the displayed entity is known to be stale). Since the protocol normally allows the user agent to determine if responses are stale or not, this indication need only be displayed when this actually happens. The indication need not be a dialog box; it could be an icon (for example, a picture of a rotting fish) or some other visual indicator.

If the user has overridden the caching mechanisms in a way that would abnormally reduce the effectiveness of caches, the user agent should continually display an indication (for example, a picture of currency in flames) so that the user does not inadvertently consume excess resources or suffer from excessive latency.

13.1.5 Exceptions to the Rules and Warnings

In some cases, the operator of a cache may choose to configure it to return stale responses even when not requested by clients. This decision should not be made lightly, but may be necessary for reasons of availability or performance, especially when the cache is poorly connected to the origin server. Whenever a cache returns a stale response, it MUST mark it as such (using a Warning header). This allows the client software to alert the user that there may be a potential problem.

It also allows the user agent to take steps to obtain a first-hand or fresh response. For this reason, a cache SHOULD NOT return a stale response if the client explicitly requests a first-hand or fresh one, unless it is impossible to comply for technical or policy reasons.

13.1.6 Client-controlled Behavior

While the origin server (and to a lesser extent, intermediate caches, by their contribution to the age of a response) are the primary source of expiration information, in some cases the client may need to control a cache's decision about whether to return a cached response without validating it. Clients do this using several directives of the Cache-Control header.

A client's request may specify the maximum age it is willing to accept of an unvalidated response; specifying a value of zero forces the cache(s) to revalidate all responses. A client may also specify the minimum time remaining before a response expires. Both of these options increase constraints on the behavior of caches, and so cannot further relax the cache's approximation of semantic transparency.

A client may also specify that it will accept stale responses, up to some maximum amount of staleness. This loosens the constraints on the caches, and so may violate the origin server's specified constraints on semantic transparency, but may be necessary to support disconnected operation, or high availability in the face of poor connectivity.

13.2 Expiration Model

13.2.1 Server-Specified Expiration

HTTP caching works best when caches can entirely avoid making requests to the origin server. The primary mechanism for avoiding requests is for an origin server to provide an explicit expiration time in the future, indicating that a response may be used to satisfy subsequent requests. In other words, a cache can return a fresh

response without first contacting the server.

Our expectation is that servers will assign future explicit expiration times to responses in the belief that the entity is not likely to change, in a semantically significant way, before the expiration time is reached. This normally preserves semantic transparency, as long as the server's expiration times are carefully chosen.

The expiration mechanism applies only to responses taken from a cache and not to first-hand responses forwarded immediately to the requesting client.

If an origin server wishes to force a semantically transparent cache to validate every request, it may assign an explicit expiration time in the past. This means that the response is always stale, and so the cache SHOULD validate it before using it for subsequent requests. See section 14.9.4 for a more restrictive way to force revalidation.

If an origin server wishes to force any HTTP/1.1 cache, no matter how it is configured, to validate every request, it should use the "must-revalidate" Cache-Control directive (see section 14.9).

Servers specify explicit expiration times using either the Expires header, or the max-age directive of the Cache-Control header.

An expiration time cannot be used to force a user agent to refresh its display or reload a resource; its semantics apply only to caching mechanisms, and such mechanisms need only check a resource's expiration status when a new request for that resource is initiated. See section 13.13 for explanation of the difference between caches and history mechanisms.

13.2.2 Heuristic Expiration

Since origin servers do not always provide explicit expiration times, HTTP caches typically assign heuristic expiration times, employing algorithms that use other header values (such as the Last-Modified time) to estimate a plausible expiration time. The HTTP/1.1 specification does not provide specific algorithms, but does impose worst-case constraints on their results. Since heuristic expiration times may compromise semantic transparency, they should be used cautiously, and we encourage origin servers to provide explicit expiration times as much as possible.

13.2.3 Age Calculations

In order to know if a cached entry is fresh, a cache needs to know if its age exceeds its freshness lifetime. We discuss how to calculate the latter in section 13.2.4; this section describes how to calculate the age of a response or cache entry.

In this discussion, we use the term "now" to mean "the current value of the clock at the host performing the calculation." Hosts that use HTTP, but especially hosts running origin servers and caches, should use NTP [28] or some similar protocol to synchronize their clocks to a globally accurate time standard.

Also note that HTTP/1.1 requires origin servers to send a Date header with every response, giving the time at which the response was generated. We use the term "date_value" to denote the value of the Date header, in a form appropriate for arithmetic operations.

HTTP/1.1 uses the Age response-header to help convey age information between caches. The Age header value is the sender's estimate of the amount of time since the response was generated at the origin server. In the case of a cached response that has been revalidated with the origin server, the Age value is based on the time of revalidation, not of the original response.

In essence, the Age value is the sum of the time that the response has been resident in each of the caches along the path from the origin server, plus the amount of time it has been in transit along network paths.

We use the term "age_value" to denote the value of the Age header, in a form appropriate for arithmetic operations.

A response's age can be calculated in two entirely independent ways:

1. now minus date_value, if the local clock is reasonably well synchronized to the origin server's clock. If the result is negative, the result is replaced by zero.

2. age_value, if all of the caches along the response path implement HTTP/1.1.

Given that we have two independent ways to compute the age of a response when it is received, we can combine these as

```
corrected_received_age = max(now - date_value, age_value)
```

and as long as we have either nearly synchronized clocks or all-

HTTP/1.1 paths, one gets a reliable (conservative) result.

Note that this correction is applied at each HTTP/1.1 cache along the path, so that if there is an HTTP/1.0 cache in the path, the correct received age is computed as long as the receiving cache's clock is nearly in sync. We don't need end-to-end clock synchronization (although it is good to have), and there is no explicit clock synchronization step.

Because of network-imposed delays, some significant interval may pass from the time that a server generates a response and the time it is received at the next outbound cache or client. If uncorrected, this delay could result in improperly low ages.

Because the request that resulted in the returned Age value must have been initiated prior to that Age value's generation, we can correct for delays imposed by the network by recording the time at which the request was initiated. Then, when an Age value is received, it MUST be interpreted relative to the time the request was initiated, not the time that the response was received. This algorithm results in conservative behavior no matter how much delay is experienced. So, we compute:

```
corrected_initial_age = corrected_received_age
                      + (now - request_time)
```

where "request_time" is the time (according to the local clock) when the request that elicited this response was sent.

Summary of age calculation algorithm, when a cache receives a response:

```
/*
 * age_value
 *      is the value of Age: header received by the cache with
 *              this response.
 * date_value
 *      is the value of the origin server's Date: header
 * request_time
 *      is the (local) time when the cache made the request
 *              that resulted in this cached response
 * response_time
 *      is the (local) time when the cache received the
 *              response
 * now
 *      is the current (local) time
 */
apparent_age = max(0, response_time - date_value);
```

```
corrected_received_age = max(apparent_age, age_value);
response_delay = response_time - request_time;
corrected_initial_age = corrected_received_age + response_delay;
resident_time = now - response_time;
current_age   = corrected_initial_age + resident_time;
```

When a cache sends a response, it must add to the corrected_initial_age the amount of time that the response was resident locally. It must then transmit this total age, using the Age header, to the next recipient cache.

Note that a client cannot reliably tell that a response is first-hand, but the presence of an Age header indicates that a response is definitely not first-hand. Also, if the Date in a response is earlier than the client's local request time, the response is probably not first-hand (in the absence of serious clock skew).

13.2.4 Expiration Calculations

In order to decide whether a response is fresh or stale, we need to compare its freshness lifetime to its age. The age is calculated as described in section 13.2.3; this section describes how to calculate the freshness lifetime, and to determine if a response has expired. In the discussion below, the values can be represented in any form appropriate for arithmetic operations.

We use the term "expires_value" to denote the value of the Expires header. We use the term "max_age_value" to denote an appropriate value of the number of seconds carried by the max-age directive of the Cache-Control header in a response (see section 14.10.

The max-age directive takes priority over Expires, so if max-age is present in a response, the calculation is simply:

```
freshness_lifetime = max_age_value
```

Otherwise, if Expires is present in the response, the calculation is:

```
freshness_lifetime = expires_value - date_value
```

Note that neither of these calculations is vulnerable to clock skew, since all of the information comes from the origin server.

If neither Expires nor Cache-Control: max-age appears in the response, and the response does not include other restrictions on caching, the cache MAY compute a freshness lifetime using a heuristic. If the value is greater than 24 hours, the cache must attach Warning 13 to any response whose age is more than 24 hours if

such warning has not already been added.

Also, if the response does have a Last-Modified time, the heuristic expiration value SHOULD be no more than some fraction of the interval since that time. A typical setting of this fraction might be 10%.

The calculation to determine if a response has expired is quite simple:

```
response_is_fresh = (freshness_lifetime > current_age)
```

13.2.5 Disambiguating Expiration Values

Because expiration values are assigned optimistically, it is possible for two caches to contain fresh values for the same resource that are different.

If a client performing a retrieval receives a non-first-hand response for a request that was already fresh in its own cache, and the Date header in its existing cache entry is newer than the Date on the new response, then the client MAY ignore the response. If so, it MAY retry the request with a "Cache-Control: max-age=0" directive (see section 14.9), to force a check with the origin server.

If a cache has two fresh responses for the same representation with different validators, it MUST use the one with the more recent Date header. This situation may arise because the cache is pooling responses from other caches, or because a client has asked for a reload or a revalidation of an apparently fresh cache entry.

13.2.6 Disambiguating Multiple Responses

Because a client may be receiving responses via multiple paths, so that some responses flow through one set of caches and other responses flow through a different set of caches, a client may receive responses in an order different from that in which the origin server sent them. We would like the client to use the most recently generated response, even if older responses are still apparently fresh.

Neither the entity tag nor the expiration value can impose an ordering on responses, since it is possible that a later response intentionally carries an earlier expiration time. However, the HTTP/1.1 specification requires the transmission of Date headers on every response, and the Date values are ordered to a granularity of one second.

When a client tries to revalidate a cache entry, and the response it receives contains a Date header that appears to be older than the one for the existing entry, then the client SHOULD repeat the request unconditionally, and include

```
Cache-Control: max-age=0
```

to force any intermediate caches to validate their copies directly with the origin server, or

```
Cache-Control: no-cache
```

to force any intermediate caches to obtain a new copy from the origin server.

If the Date values are equal, then the client may use either response (or may, if it is being extremely prudent, request a new response). Servers MUST NOT depend on clients being able to choose deterministically between responses generated during the same second, if their expiration times overlap.

13.3 Validation Model

When a cache has a stale entry that it would like to use as a response to a client's request, it first has to check with the origin server (or possibly an intermediate cache with a fresh response) to see if its cached entry is still usable. We call this "validating" the cache entry. Since we do not want to have to pay the overhead of retransmitting the full response if the cached entry is good, and we do not want to pay the overhead of an extra round trip if the cached entry is invalid, the HTTP/1.1 protocol supports the use of conditional methods.

The key protocol features for supporting conditional methods are those concerned with "cache validators." When an origin server generates a full response, it attaches some sort of validator to it, which is kept with the cache entry. When a client (user agent or proxy cache) makes a conditional request for a resource for which it has a cache entry, it includes the associated validator in the request.

The server then checks that validator against the current validator for the entity, and, if they match, it responds with a special status code (usually, 304 (Not Modified)) and no entity-body. Otherwise, it returns a full response (including entity-body). Thus, we avoid transmitting the full response if the validator matches, and we avoid an extra round trip if it does not match.

Note: the comparison functions used to decide if validators match are defined in section 13.3.3.

In HTTP/1.1, a conditional request looks exactly the same as a normal request for the same resource, except that it carries a special header (which includes the validator) that implicitly turns the method (usually, GET) into a conditional.

The protocol includes both positive and negative senses of cache-validating conditions. That is, it is possible to request either that a method be performed if and only if a validator matches or if and only if no validators match.

Note: a response that lacks a validator may still be cached, and served from cache until it expires, unless this is explicitly prohibited by a Cache-Control directive. However, a cache cannot do a conditional retrieval if it does not have a validator for the entity, which means it will not be refreshable after it expires.

13.3.1 Last-modified Dates

The Last-Modified entity-header field value is often used as a cache validator. In simple terms, a cache entry is considered to be valid if the entity has not been modified since the Last-Modified value.

13.3.2 Entity Tag Cache Validators

The ETag entity-header field value, an entity tag, provides for an "opaque" cache validator. This may allow more reliable validation in situations where it is inconvenient to store modification dates, where the one-second resolution of HTTP date values is not sufficient, or where the origin server wishes to avoid certain paradoxes that may arise from the use of modification dates.

Entity Tags are described in section 3.11. The headers used with entity tags are described in sections 14.20, 14.25, 14.26 and 14.43.

13.3.3 Weak and Strong Validators

Since both origin servers and caches will compare two validators to decide if they represent the same or different entities, one normally would expect that if the entity (the entity-body or any entity-headers) changes in any way, then the associated validator would change as well. If this is true, then we call this validator a "strong validator."

However, there may be cases when a server prefers to change the validator only on semantically significant changes, and not when

insignificant aspects of the entity change. A validator that does not always change when the resource changes is a "weak validator."

Entity tags are normally "strong validators," but the protocol provides a mechanism to tag an entity tag as "weak." One can think of a strong validator as one that changes whenever the bits of an entity changes, while a weak value changes whenever the meaning of an entity changes. Alternatively, one can think of a strong validator as part of an identifier for a specific entity, while a weak validator is part of an identifier for a set of semantically equivalent entities.

Note: One example of a strong validator is an integer that is incremented in stable storage every time an entity is changed.

An entity's modification time, if represented with one-second resolution, could be a weak validator, since it is possible that the resource may be modified twice during a single second.

Support for weak validators is optional; however, weak validators allow for more efficient caching of equivalent objects; for example, a hit counter on a site is probably good enough if it is updated every few days or weeks, and any value during that period is likely "good enough" to be equivalent.

A "use" of a validator is either when a client generates a request and includes the validator in a validating header field, or when a server compares two validators.

Strong validators are usable in any context. Weak validators are only usable in contexts that do not depend on exact equality of an entity. For example, either kind is usable for a conditional GET of a full entity. However, only a strong validator is usable for a sub-range retrieval, since otherwise the client may end up with an internally inconsistent entity.

The only function that the HTTP/1.1 protocol defines on validators is comparison. There are two validator comparison functions, depending on whether the comparison context allows the use of weak validators or not:

- The strong comparison function: in order to be considered equal, both validators must be identical in every way, and neither may be weak.
- The weak comparison function: in order to be considered equal, both validators must be identical in every way, but either or both of them may be tagged as "weak" without affecting the result.

The weak comparison function MAY be used for simple (non-subrange)

GET requests. The strong comparison function MUST be used in all other cases.

An entity tag is strong unless it is explicitly tagged as weak. Section 3.11 gives the syntax for entity tags.

A Last-Modified time, when used as a validator in a request, is implicitly weak unless it is possible to deduce that it is strong, using the following rules:

- The validator is being compared by an origin server to the actual current validator for the entity and,
- That origin server reliably knows that the associated entity did not change twice during the second covered by the presented validator.

or

- The validator is about to be used by a client in an If-Modified-Since or If-Unmodified-Since header, because the client has a cache entry for the associated entity, and
- That cache entry includes a Date value, which gives the time when the origin server sent the original response, and
- The presented Last-Modified time is at least 60 seconds before the Date value.

or

- The validator is being compared by an intermediate cache to the validator stored in its cache entry for the entity, and
- That cache entry includes a Date value, which gives the time when the origin server sent the original response, and
- The presented Last-Modified time is at least 60 seconds before the Date value.

This method relies on the fact that if two different responses were sent by the origin server during the same second, but both had the same Last-Modified time, then at least one of those responses would have a Date value equal to its Last-Modified time. The arbitrary 60-second limit guards against the possibility that the Date and Last-Modified values are generated from different clocks, or at somewhat different times during the preparation of the response. An implementation may use a value larger than 60 seconds, if it is believed that 60 seconds is too short.

If a client wishes to perform a sub-range retrieval on a value for which it has only a Last-Modified time and no opaque validator, it may do this only if the Last-Modified time is strong in the sense described here.

A cache or origin server receiving a cache-conditional request, other than a full-body GET request, MUST use the strong comparison function to evaluate the condition.

These rules allow HTTP/1.1 caches and clients to safely perform sub-range retrievals on values that have been obtained from HTTP/1.0 servers.

13.3.4 Rules for When to Use Entity Tags and Last-modified Dates

We adopt a set of rules and recommendations for origin servers, clients, and caches regarding when various validator types should be used, and for what purposes.

HTTP/1.1 origin servers:

- SHOULD send an entity tag validator unless it is not feasible to generate one.
- MAY send a weak entity tag instead of a strong entity tag, if performance considerations support the use of weak entity tags, or if it is unfeasible to send a strong entity tag.
- SHOULD send a Last-Modified value if it is feasible to send one, unless the risk of a breakdown in semantic transparency that could result from using this date in an If-Modified-Since header would lead to serious problems.

In other words, the preferred behavior for an HTTP/1.1 origin server is to send both a strong entity tag and a Last-Modified value.

In order to be legal, a strong entity tag MUST change whenever the associated entity value changes in any way. A weak entity tag SHOULD change whenever the associated entity changes in a semantically significant way.

> Note: in order to provide semantically transparent caching, an origin server must avoid reusing a specific strong entity tag value for two different entities, or reusing a specific weak entity tag value for two semantically different entities. Cache entries may persist for arbitrarily long periods, regardless of expiration times, so it may be inappropriate to expect that a cache will never again attempt to validate an entry using a validator that it obtained at some point in the past.

HTTP/1.1 clients:

- If an entity tag has been provided by the origin server, MUST use that entity tag in any cache-conditional request (using If-Match or If-None-Match).
- If only a Last-Modified value has been provided by the origin server, SHOULD use that value in non-subrange cache-conditional requests (using If-Modified-Since).
- If only a Last-Modified value has been provided by an HTTP/1.0 origin server, MAY use that value in subrange cache-conditional requests (using If-Unmodified-Since:). The user agent should provide a way to disable this, in case of difficulty.
- If both an entity tag and a Last-Modified value have been provided by the origin server, SHOULD use both validators in cache-conditional requests. This allows both HTTP/1.0 and HTTP/1.1 caches to respond appropriately.

An HTTP/1.1 cache, upon receiving a request, MUST use the most restrictive validator when deciding whether the client's cache entry matches the cache's own cache entry. This is only an issue when the request contains both an entity tag and a last-modified-date validator (If-Modified-Since or If-Unmodified-Since).

> A note on rationale: The general principle behind these rules is that HTTP/1.1 servers and clients should transmit as much non-redundant information as is available in their responses and requests. HTTP/1.1 systems receiving this information will make the most conservative assumptions about the validators they receive.
>
> HTTP/1.0 clients and caches will ignore entity tags. Generally, last-modified values received or used by these systems will support transparent and efficient caching, and so HTTP/1.1 origin servers should provide Last-Modified values. In those rare cases where the use of a Last-Modified value as a validator by an HTTP/1.0 system could result in a serious problem, then HTTP/1.1 origin servers should not provide one.

13.3.5 Non-validating Conditionals

The principle behind entity tags is that only the service author knows the semantics of a resource well enough to select an appropriate cache validation mechanism, and the specification of any validator comparison function more complex than byte-equality would open up a can of worms. Thus, comparisons of any other headers (except Last-Modified, for compatibility with HTTP/1.0) are never used for purposes of validating a cache entry.

13.4 Response Cachability

Unless specifically constrained by a Cache-Control (section 14.9) directive, a caching system may always store a successful response (see section 13.8) as a cache entry, may return it without validation if it is fresh, and may return it after successful validation. If

there is neither a cache validator nor an explicit expiration time associated with a response, we do not expect it to be cached, but certain caches may violate this expectation (for example, when little or no network connectivity is available). A client can usually detect that such a response was taken from a cache by comparing the Date header to the current time.

Note that some HTTP/1.0 caches are known to violate this expectation without providing any Warning.

However, in some cases it may be inappropriate for a cache to retain an entity, or to return it in response to a subsequent request. This may be because absolute semantic transparency is deemed necessary by the service author, or because of security or privacy considerations. Certain Cache-Control directives are therefore provided so that the server can indicate that certain resource entities, or portions thereof, may not be cached regardless of other considerations.

Note that section 14.8 normally prevents a shared cache from saving and returning a response to a previous request if that request included an Authorization header.

A response received with a status code of 200, 203, 206, 300, 301 or 410 may be stored by a cache and used in reply to a subsequent request, subject to the expiration mechanism, unless a Cache-Control directive prohibits caching. However, a cache that does not support the Range and Content-Range headers MUST NOT cache 206 (Partial Content) responses.

A response received with any other status code MUST NOT be returned in a reply to a subsequent request unless there are Cache-Control directives or another header(s) that explicitly allow it. For example, these include the following: an Expires header (section 14.21); a "max-age", "must-revalidate", "proxy-revalidate", "public" or "private" Cache-Control directive (section 14.9).

13.5 Constructing Responses From Caches

The purpose of an HTTP cache is to store information received in response to requests, for use in responding to future requests. In many cases, a cache simply returns the appropriate parts of a response to the requester. However, if the cache holds a cache entry based on a previous response, it may have to combine parts of a new response with what is held in the cache entry.

13.5.1 End-to-end and Hop-by-hop Headers

For the purpose of defining the behavior of caches and non-caching proxies, we divide HTTP headers into two categories:

- End-to-end headers, which must be transmitted to the ultimate recipient of a request or response. End-to-end headers in responses must be stored as part of a cache entry and transmitted in any response formed from a cache entry.
- Hop-by-hop headers, which are meaningful only for a single transport-level connection, and are not stored by caches or forwarded by proxies.

The following HTTP/1.1 headers are hop-by-hop headers:

- Connection
- Keep-Alive
- Public
- Proxy-Authenticate
- Transfer-Encoding
- Upgrade

All other headers defined by HTTP/1.1 are end-to-end headers.

Hop-by-hop headers introduced in future versions of HTTP MUST be listed in a Connection header, as described in section 14.10.

13.5.2 Non-modifiable Headers

Some features of the HTTP/1.1 protocol, such as Digest Authentication, depend on the value of certain end-to-end headers. A cache or non-caching proxy SHOULD NOT modify an end-to-end header unless the definition of that header requires or specifically allows that.

A cache or non-caching proxy MUST NOT modify any of the following fields in a request or response, nor may it add any of these fields if not already present:

- Content-Location
- ETag
- Expires
- Last-Modified

A cache or non-caching proxy MUST NOT modify or add any of the following fields in a response that contains the no-transform Cache-Control directive, or in any request:

- Content-Encoding
- Content-Length
- Content-Range
- Content-Type

A cache or non-caching proxy MAY modify or add these fields in a response that does not include no-transform, but if it does so, it MUST add a Warning 14 (Transformation applied) if one does not already appear in the response.

> Warning: unnecessary modification of end-to-end headers may cause authentication failures if stronger authentication mechanisms are introduced in later versions of HTTP. Such authentication mechanisms may rely on the values of header fields not listed here.

13.5.3 Combining Headers

When a cache makes a validating request to a server, and the server provides a 304 (Not Modified) response, the cache must construct a response to send to the requesting client. The cache uses the entity-body stored in the cache entry as the entity-body of this outgoing response. The end-to-end headers stored in the cache entry are used for the constructed response, except that any end-to-end headers provided in the 304 response MUST replace the corresponding headers from the cache entry. Unless the cache decides to remove the cache entry, it MUST also replace the end-to-end headers stored with the cache entry with corresponding headers received in the incoming response.

In other words, the set of end-to-end headers received in the incoming response overrides all corresponding end-to-end headers stored with the cache entry. The cache may add Warning headers (see section 14.45) to this set.

If a header field-name in the incoming response matches more than one header in the cache entry, all such old headers are replaced.

> Note: this rule allows an origin server to use a 304 (Not Modified) response to update any header associated with a previous response for the same entity, although it might not always be meaningful or correct to do so. This rule does not allow an origin server to use a 304 (not Modified) response to entirely delete a header that it had provided with a previous response.

13.5.4 Combining Byte Ranges

A response may transfer only a subrange of the bytes of an entity-body, either because the request included one or more Range specifications, or because a connection was broken prematurely. After several such transfers, a cache may have received several ranges of the same entity-body.

If a cache has a stored non-empty set of subranges for an entity, and an incoming response transfers another subrange, the cache MAY combine the new subrange with the existing set if both the following conditions are met:

- Both the incoming response and the cache entry must have a cache validator.
- The two cache validators must match using the strong comparison function (see section 13.3.3).

If either requirement is not meant, the cache must use only the most recent partial response (based on the Date values transmitted with every response, and using the incoming response if these values are equal or missing), and must discard the other partial information.

13.6 Caching Negotiated Responses

Use of server-driven content negotiation (section 12), as indicated by the presence of a Vary header field in a response, alters the conditions and procedure by which a cache can use the response for subsequent requests.

A server MUST use the Vary header field (section 14.43) to inform a cache of what header field dimensions are used to select among multiple representations of a cachable response. A cache may use the selected representation (the entity included with that particular response) for replying to subsequent requests on that resource only when the subsequent requests have the same or equivalent values for all header fields specified in the Vary response-header. Requests with a different value for one or more of those header fields would be forwarded toward the origin server.

If an entity tag was assigned to the representation, the forwarded request SHOULD be conditional and include the entity tags in an If-None-Match header field from all its cache entries for the Request-URI. This conveys to the server the set of entities currently held by the cache, so that if any one of these entities matches the requested entity, the server can use the ETag header in its 304 (Not Modified) response to tell the cache which entry is appropriate. If the entity-tag of the new response matches that of an existing entry, the

new response SHOULD be used to update the header fields of the existing entry, and the result MUST be returned to the client.

The Vary header field may also inform the cache that the representation was selected using criteria not limited to the request-headers; in this case, a cache MUST NOT use the response in a reply to a subsequent request unless the cache relays the new request to the origin server in a conditional request and the server responds with 304 (Not Modified), including an entity tag or Content-Location that indicates which entity should be used.

If any of the existing cache entries contains only partial content for the associated entity, its entity-tag SHOULD NOT be included in the If-None-Match header unless the request is for a range that would be fully satisfied by that entry.

If a cache receives a successful response whose Content-Location field matches that of an existing cache entry for the same Request-URI, whose entity-tag differs from that of the existing entry, and whose Date is more recent than that of the existing entry, the existing entry SHOULD NOT be returned in response to future requests, and should be deleted from the cache.

13.7 Shared and Non-Shared Caches

For reasons of security and privacy, it is necessary to make a distinction between "shared" and "non-shared" caches. A non-shared cache is one that is accessible only to a single user. Accessibility in this case SHOULD be enforced by appropriate security mechanisms. All other caches are considered to be "shared." Other sections of this specification place certain constraints on the operation of shared caches in order to prevent loss of privacy or failure of access controls.

13.8 Errors or Incomplete Response Cache Behavior

A cache that receives an incomplete response (for example, with fewer bytes of data than specified in a Content-Length header) may store the response. However, the cache MUST treat this as a partial response. Partial responses may be combined as described in section 13.5.4; the result might be a full response or might still be partial. A cache MUST NOT return a partial response to a client without explicitly marking it as such, using the 206 (Partial Content) status code. A cache MUST NOT return a partial response using a status code of 200 (OK).

If a cache receives a 5xx response while attempting to revalidate an entry, it may either forward this response to the requesting client,

or act as if the server failed to respond. In the latter case, it MAY return a previously received response unless the cached entry includes the "must-revalidate" Cache-Control directive (see section 14.9).

13.9 Side Effects of GET and HEAD

Unless the origin server explicitly prohibits the caching of their responses, the application of GET and HEAD methods to any resources SHOULD NOT have side effects that would lead to erroneous behavior if these responses are taken from a cache. They may still have side effects, but a cache is not required to consider such side effects in its caching decisions. Caches are always expected to observe an origin server's explicit restrictions on caching.

We note one exception to this rule: since some applications have traditionally used GETs and HEADs with query URLs (those containing a "?" in the rel_path part) to perform operations with significant side effects, caches MUST NOT treat responses to such URLs as fresh unless the server provides an explicit expiration time. This specifically means that responses from HTTP/1.0 servers for such URIs should not be taken from a cache. See section 9.1.1 for related information.

13.10 Invalidation After Updates or Deletions

The effect of certain methods at the origin server may cause one or more existing cache entries to become non-transparently invalid. That is, although they may continue to be "fresh," they do not accurately reflect what the origin server would return for a new request.

There is no way for the HTTP protocol to guarantee that all such cache entries are marked invalid. For example, the request that caused the change at the origin server may not have gone through the proxy where a cache entry is stored. However, several rules help reduce the likelihood of erroneous behavior.

In this section, the phrase "invalidate an entity" means that the cache should either remove all instances of that entity from its storage, or should mark these as "invalid" and in need of a mandatory revalidation before they can be returned in response to a subsequent request.

Some HTTP methods may invalidate an entity. This is either the entity referred to by the Request-URI, or by the Location or Content-Location response-headers (if present). These methods are:

- o PUT
- o DELETE
- o POST

In order to prevent denial of service attacks, an invalidation based on the URI in a Location or Content-Location header MUST only be performed if the host part is the same as in the Request-URI.

13.11 Write-Through Mandatory

All methods that may be expected to cause modifications to the origin server's resources MUST be written through to the origin server. This currently includes all methods except for GET and HEAD. A cache MUST NOT reply to such a request from a client before having transmitted the request to the inbound server, and having received a corresponding response from the inbound server. This does not prevent a cache from sending a 100 (Continue) response before the inbound server has replied.

The alternative (known as "write-back" or "copy-back" caching) is not allowed in HTTP/1.1, due to the difficulty of providing consistent updates and the problems arising from server, cache, or network failure prior to write-back.

13.12 Cache Replacement

If a new cachable (see sections 14.9.2, 13.2.5, 13.2.6 and 13.8) response is received from a resource while any existing responses for the same resource are cached, the cache SHOULD use the new response to reply to the current request. It may insert it into cache storage and may, if it meets all other requirements, use it to respond to any future requests that would previously have caused the old response to be returned. If it inserts the new response into cache storage it should follow the rules in section 13.5.3.

> Note: a new response that has an older Date header value than existing cached responses is not cachable.

13.13 History Lists

User agents often have history mechanisms, such as "Back" buttons and history lists, which can be used to redisplay an entity retrieved earlier in a session.

History mechanisms and caches are different. In particular history mechanisms SHOULD NOT try to show a semantically transparent view of the current state of a resource. Rather, a history mechanism is meant to show exactly what the user saw at the time when the resource was retrieved.

By default, an expiration time does not apply to history mechanisms. If the entity is still in storage, a history mechanism should display it even if the entity has expired, unless the user has specifically configured the agent to refresh expired history documents.

This should not be construed to prohibit the history mechanism from telling the user that a view may be stale.

> Note: if history list mechanisms unnecessarily prevent users from viewing stale resources, this will tend to force service authors to avoid using HTTP expiration controls and cache controls when they would otherwise like to. Service authors may consider it important that users not be presented with error messages or warning messages when they use navigation controls (such as BACK) to view previously fetched resources. Even though sometimes such resources ought not to cached, or ought to expire quickly, user interface considerations may force service authors to resort to other means of preventing caching (e.g. "once-only" URLs) in order not to suffer the effects of improperly functioning history mechanisms.

14 Header Field Definitions

This section defines the syntax and semantics of all standard HTTP/1.1 header fields. For entity-header fields, both sender and recipient refer to either the client or the server, depending on who sends and who receives the entity.

14.1 Accept

The Accept request-header field can be used to specify certain media types which are acceptable for the response. Accept headers can be used to indicate that the request is specifically limited to a small set of desired types, as in the case of a request for an in-line image.

```
Accept         = "Accept" ":"
                 #( media-range [ accept-params ] )

media-range    = ( "*/*"
                 | ( type "/" "*" )
                 | ( type "/" subtype )
                 ) *( ";" parameter )

accept-params  = ";" "q" "=" qvalue *( accept-extension )

accept-extension = ";" token [ "=" ( token | quoted-string ) ]
```

The asterisk "*" character is used to group media types into ranges, with "*/*" indicating all media types and "type/*" indicating all subtypes of that type. The media-range MAY include media type parameters that are applicable to that range.

Each media-range MAY be followed by one or more accept-params, beginning with the "q" parameter for indicating a relative quality factor. The first "q" parameter (if any) separates the media-range parameter(s) from the accept-params. Quality factors allow the user or user agent to indicate the relative degree of preference for that media-range, using the qvalue scale from 0 to 1 (section 3.9). The default value is q=1.

> Note: Use of the "q" parameter name to separate media type parameters from Accept extension parameters is due to historical practice. Although this prevents any media type parameter named "q" from being used with a media range, such an event is believed to be unlikely given the lack of any "q" parameters in the IANA media type registry and the rare usage of any media type parameters in Accept. Future media types should be discouraged from registering any parameter named "q".

The example

```
Accept: audio/*; q=0.2, audio/basic
```

SHOULD be interpreted as "I prefer audio/basic, but send me any audio type if it is the best available after an 80% mark-down in quality."

If no Accept header field is present, then it is assumed that the client accepts all media types. If an Accept header field is present, and if the server cannot send a response which is acceptable according to the combined Accept field value, then the server SHOULD send a 406 (not acceptable) response.

A more elaborate example is

```
Accept: text/plain; q=0.5, text/html,
        text/x-dvi; q=0.8, text/x-c
```

Verbally, this would be interpreted as "text/html and text/x-c are the preferred media types, but if they do not exist, then send the text/x-dvi entity, and if that does not exist, send the text/plain entity."

Media ranges can be overridden by more specific media ranges or specific media types. If more than one media range applies to a given type, the most specific reference has precedence. For example,

```
Accept: text/*, text/html, text/html;level=1, */*
```

have the following precedence:

```
1) text/html;level=1
2) text/html
3) text/*
4) */*
```

The media type quality factor associated with a given type is determined by finding the media range with the highest precedence which matches that type. For example,

```
Accept: text/*;q=0.3, text/html;q=0.7, text/html;level=1,
        text/html;level=2;q=0.4, */*;q=0.5
```

would cause the following values to be associated:

```
text/html;level=1         = 1
text/html                 = 0.7
text/plain                = 0.3
image/jpeg                = 0.5
text/html;level=2         = 0.4
text/html;level=3         = 0.7
```

> Note: A user agent may be provided with a default set of quality values for certain media ranges. However, unless the user agent is a closed system which cannot interact with other rendering agents,

this default set should be configurable by the user.

14.2 Accept-Charset

The Accept-Charset request-header field can be used to indicate what character sets are acceptable for the response. This field allows clients capable of understanding more comprehensive or special-purpose character sets to signal that capability to a server which is capable of representing documents in those character sets. The ISO-8859-1 character set can be assumed to be acceptable to all user agents.

```
Accept-Charset = "Accept-Charset" ":"
          1#( charset [ ";" "q" "=" qvalue ] )
```

Character set values are described in section 3.4. Each charset may be given an associated quality value which represents the user's preference for that charset. The default value is q=1. An example is

```
Accept-Charset: iso-8859-5, unicode-1-1;q=0.8
```

If no Accept-Charset header is present, the default is that any character set is acceptable. If an Accept-Charset header is present, and if the server cannot send a response which is acceptable according to the Accept-Charset header, then the server SHOULD send an error response with the 406 (not acceptable) status code, though the sending of an unacceptable response is also allowed.

14.3 Accept-Encoding

The Accept-Encoding request-header field is similar to Accept, but restricts the content-coding values (section 14.12) which are acceptable in the response.

```
Accept-Encoding  = "Accept-Encoding" ":"
                           #( content-coding )
```

An example of its use is

```
Accept-Encoding: compress, gzip
```

If no Accept-Encoding header is present in a request, the server MAY assume that the client will accept any content coding. If an Accept-Encoding header is present, and if the server cannot send a response which is acceptable according to the Accept-Encoding header, then the server SHOULD send an error response with the 406 (Not Acceptable) status code.

An empty Accept-Encoding value indicates none are acceptable.

14.4 Accept-Language

The Accept-Language request-header field is similar to Accept, but restricts the set of natural languages that are preferred as a response to the request.

```
Accept-Language = "Accept-Language" ":"
                  1#( language-range [ ";" "q" "=" qvalue ] )

language-range  = ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) | "*" )
```

Each language-range MAY be given an associated quality value which represents an estimate of the user's preference for the languages specified by that range. The quality value defaults to "q=1". For example,

```
Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

would mean: "I prefer Danish, but will accept British English and other types of English." A language-range matches a language-tag if it exactly equals the tag, or if it exactly equals a prefix of the tag such that the first tag character following the prefix is "-". The special range "*", if present in the Accept-Language field, matches every tag not matched by any other range present in the Accept-Language field.

> Note: This use of a prefix matching rule does not imply that language tags are assigned to languages in such a way that it is always true that if a user understands a language with a certain tag, then this user will also understand all languages with tags for which this tag is a prefix. The prefix rule simply allows the use of prefix tags if this is the case.

The language quality factor assigned to a language-tag by the Accept-Language field is the quality value of the longest language-range in the field that matches the language-tag. If no language-range in the field matches the tag, the language quality factor assigned is 0. If no Accept-Language header is present in the request, the server SHOULD assume that all languages are equally acceptable. If an Accept-Language header is present, then all languages which are assigned a quality factor greater than 0 are acceptable.

It may be contrary to the privacy expectations of the user to send an Accept-Language header with the complete linguistic preferences of the user in every request. For a discussion of this issue, see

section 15.7.

Note: As intelligibility is highly dependent on the individual user, it is recommended that client applications make the choice of linguistic preference available to the user. If the choice is not made available, then the Accept-Language header field must not be given in the request.

14.5 Accept-Ranges

The Accept-Ranges response-header field allows the server to indicate its acceptance of range requests for a resource:

```
Accept-Ranges     = "Accept-Ranges" ":" acceptable-ranges

acceptable-ranges = 1#range-unit | "none"
```

Origin servers that accept byte-range requests MAY send

```
Accept-Ranges: bytes
```

but are not required to do so. Clients MAY generate byte-range requests without having received this header for the resource involved.

Servers that do not accept any kind of range request for a resource MAY send

```
Accept-Ranges: none
```

to advise the client not to attempt a range request.

14.6 Age

The Age response-header field conveys the sender's estimate of the amount of time since the response (or its revalidation) was generated at the origin server. A cached response is "fresh" if its age does not exceed its freshness lifetime. Age values are calculated as specified in section 13.2.3.

```
Age = "Age" ":" age-value

age-value = delta-seconds
```

Age values are non-negative decimal integers, representing time in seconds.

If a cache receives a value larger than the largest positive integer it can represent, or if any of its age calculations overflows, it MUST transmit an Age header with a value of 2147483648 (2^31). HTTP/1.1 caches MUST send an Age header in every response. Caches SHOULD use an arithmetic type of at least 31 bits of range.

14.7 Allow

The Allow entity-header field lists the set of methods supported by the resource identified by the Request-URI. The purpose of this field is strictly to inform the recipient of valid methods associated with the resource. An Allow header field MUST be present in a 405 (Method Not Allowed) response.

```
Allow          = "Allow" ":" 1#method
```

Example of use:

```
Allow: GET, HEAD, PUT
```

This field cannot prevent a client from trying other methods. However, the indications given by the Allow header field value SHOULD be followed. The actual set of allowed methods is defined by the origin server at the time of each request.

The Allow header field MAY be provided with a PUT request to recommend the methods to be supported by the new or modified resource. The server is not required to support these methods and SHOULD include an Allow header in the response giving the actual supported methods.

A proxy MUST NOT modify the Allow header field even if it does not understand all the methods specified, since the user agent MAY have other means of communicating with the origin server.

The Allow header field does not indicate what methods are implemented at the server level. Servers MAY use the Public response-header field (section 14.35) to describe what methods are implemented on the server as a whole.

14.8 Authorization

A user agent that wishes to authenticate itself with a server--usually, but not necessarily, after receiving a 401 response--MAY do so by including an Authorization request-header field with the request. The Authorization field value consists of credentials containing the authentication information of the user agent for the realm of the resource being requested.

```
Authorization  = "Authorization" ":" credentials
```

HTTP access authentication is described in section 11. If a request is authenticated and a realm specified, the same credentials SHOULD be valid for all other requests within this realm.

When a shared cache (see section 13.7) receives a request containing an Authorization field, it MUST NOT return the corresponding response as a reply to any other request, unless one of the following specific exceptions holds:

1. If the response includes the "proxy-revalidate" Cache-Control directive, the cache MAY use that response in replying to a subsequent request, but a proxy cache MUST first revalidate it with the origin server, using the request-headers from the new request to allow the origin server to authenticate the new request.
2. If the response includes the "must-revalidate" Cache-Control directive, the cache MAY use that response in replying to a subsequent request, but all caches MUST first revalidate it with the origin server, using the request-headers from the new request to allow the origin server to authenticate the new request.
3. If the response includes the "public" Cache-Control directive, it may be returned in reply to any subsequent request.

14.9 Cache-Control

The Cache-Control general-header field is used to specify directives that MUST be obeyed by all caching mechanisms along the request/response chain. The directives specify behavior intended to prevent caches from adversely interfering with the request or response. These directives typically override the default caching algorithms. Cache directives are unidirectional in that the presence of a directive in a request does not imply that the same directive should be given in the response.

> Note that HTTP/1.0 caches may not implement Cache-Control and may only implement Pragma: no-cache (see section 14.32).

Cache directives must be passed through by a proxy or gateway application, regardless of their significance to that application, since the directives may be applicable to all recipients along the request/response chain. It is not possible to specify a cache-directive for a specific cache.

```
Cache-Control   = "Cache-Control" ":" 1#cache-directive

cache-directive = cache-request-directive
                | cache-response-directive
```



```
cache-request-directive =
                  "no-cache" [ "=" <"> 1#field-name <"> ]
                | "no-store"
                | "max-age" "=" delta-seconds
                | "max-stale" [ "=" delta-seconds ]
                | "min-fresh" "=" delta-seconds
                | "only-if-cached"
                | cache-extension

cache-response-directive =
                  "public"
                | "private" [ "=" <"> 1#field-name <"> ]
                | "no-cache" [ "=" <"> 1#field-name <"> ]
                | "no-store"
                | "no-transform"
                | "must-revalidate"
                | "proxy-revalidate"
                | "max-age" "=" delta-seconds
                | cache-extension

cache-extension = token [ "=" ( token | quoted-string ) ]
```

When a directive appears without any 1#field-name parameter, the directive applies to the entire request or response. When such a directive appears with a 1#field-name parameter, it applies only to the named field or fields, and not to the rest of the request or response. This mechanism supports extensibility; implementations of future versions of the HTTP protocol may apply these directives to header fields not defined in HTTP/1.1.

The cache-control directives can be broken down into these general categories:

- Restrictions on what is cachable; these may only be imposed by the origin server.
- Restrictions on what may be stored by a cache; these may be imposed by either the origin server or the user agent.
- Modifications of the basic expiration mechanism; these may be imposed by either the origin server or the user agent.
- Controls over cache revalidation and reload; these may only be imposed by a user agent.
- Control over transformation of entities.
- Extensions to the caching system.

14.9.1 What is Cachable

By default, a response is cachable if the requirements of the request method, request header fields, and the response status indicate that it is cachable. Section 13.4 summarizes these defaults for cachability. The following Cache-Control response directives allow an origin server to override the default cachability of a response:

public
: Indicates that the response is cachable by any cache, even if it would normally be non-cachable or cachable only within a non-shared cache. (See also Authorization, section 14.8, for additional details.)

private
: Indicates that all or part of the response message is intended for a single user and MUST NOT be cached by a shared cache. This allows an origin server to state that the specified parts of the response are intended for only one user and are not a valid response for requests by other users. A private (non-shared) cache may cache the response.

  Note: This usage of the word private only controls where the response may be cached, and cannot ensure the privacy of the message content.

no-cache
: Indicates that all or part of the response message MUST NOT be cached anywhere. This allows an origin server to prevent caching even by caches that have been configured to return stale responses to client requests.

  Note: Most HTTP/1.0 caches will not recognize or obey this directive.

14.9.2 What May be Stored by Caches

The purpose of the no-store directive is to prevent the inadvertent release or retention of sensitive information (for example, on backup tapes). The no-store directive applies to the entire message, and may be sent either in a response or in a request. If sent in a request, a cache MUST NOT store any part of either this request or any response to it. If sent in a response, a cache MUST NOT store any part of either this response or the request that elicited it. This directive applies to both non-shared and shared caches. "MUST NOT store" in this context means that the cache MUST NOT intentionally store the information in non-volatile storage, and MUST make a best-effort attempt to remove the information from volatile storage as promptly as possible after forwarding it.

Even when this directive is associated with a response, users may explicitly store such a response outside of the caching system (e.g., with a "Save As" dialog). History buffers may store such responses as part of their normal operation.

The purpose of this directive is to meet the stated requirements of certain users and service authors who are concerned about accidental releases of information via unanticipated accesses to cache data structures. While the use of this directive may improve privacy in some cases, we caution that it is NOT in any way a reliable or sufficient mechanism for ensuring privacy. In particular, malicious or compromised caches may not recognize or obey this directive; and communications networks may be vulnerable to eavesdropping.

14.9.3 Modifications of the Basic Expiration Mechanism

The expiration time of an entity may be specified by the origin server using the Expires header (see section 14.21). Alternatively, it may be specified using the max-age directive in a response.

If a response includes both an Expires header and a max-age directive, the max-age directive overrides the Expires header, even if the Expires header is more restrictive. This rule allows an origin server to provide, for a given response, a longer expiration time to an HTTP/1.1 (or later) cache than to an HTTP/1.0 cache. This may be useful if certain HTTP/1.0 caches improperly calculate ages or expiration times, perhaps due to desynchronized clocks.

  Note: most older caches, not compliant with this specification, do not implement any Cache-Control directives. An origin server wishing to use a Cache-Control directive that restricts, but does not prevent, caching by an HTTP/1.1-compliant cache may exploit the requirement that the max-age directive overrides the Expires header, and the fact that non-HTTP/1.1-compliant caches do not observe the max-age directive.

Other directives allow an user agent to modify the basic expiration mechanism. These directives may be specified on a request:

max-age
: Indicates that the client is willing to accept a response whose age is no greater than the specified time in seconds. Unless max-stale directive is also included, the client is not willing to accept a stale response.

min-fresh
: Indicates that the client is willing to accept a response whose freshness lifetime is no less than its current age plus the

specified time in seconds. That is, the client wants a response that will still be fresh for at least the specified number of seconds.

max-stale
Indicates that the client is willing to accept a response that has exceeded its expiration time. If max-stale is assigned a value, then the client is willing to accept a response that has exceeded its expiration time by no more than the specified number of seconds. If no value is assigned to max-stale, then the client is willing to accept a stale response of any age.

If a cache returns a stale response, either because of a max-stale directive on a request, or because the cache is configured to override the expiration time of a response, the cache MUST attach a Warning header to the stale response, using Warning 10 (Response is stale).

14.9.4 Cache Revalidation and Reload Controls

Sometimes an user agent may want or need to insist that a cache revalidate its cache entry with the origin server (and not just with the next cache along the path to the origin server), or to reload its cache entry from the origin server. End-to-end revalidation may be necessary if either the cache or the origin server has overestimated the expiration time of the cached response. End-to-end reload may be necessary if the cache entry has become corrupted for some reason.

End-to-end revalidation may be requested either when the client does not have its own local cached copy, in which case we call it "unspecified end-to-end revalidation", or when the client does have a local cached copy, in which case we call it "specific end-to-end revalidation."

The client can specify these three kinds of action using Cache-Control request directives:

End-to-end reload
The request includes a "no-cache" Cache-Control directive or, for compatibility with HTTP/1.0 clients, "Pragma: no-cache". No field names may be included with the no-cache directive in a request. The server MUST NOT use a cached copy when responding to such a request.

Specific end-to-end revalidation
The request includes a "max-age=0" Cache-Control directive, which forces each cache along the path to the origin server to revalidate its own entry, if any, with the next cache or server. The initial

request includes a cache-validating conditional with the client's current validator.

Unspecified end-to-end revalidation
The request includes "max-age=0" Cache-Control directive, which forces each cache along the path to the origin server to revalidate its own entry, if any, with the next cache or server. The initial request does not include a cache-validating conditional; the first cache along the path (if any) that holds a cache entry for this resource includes a cache-validating conditional with its current validator.

When an intermediate cache is forced, by means of a max-age=0 directive, to revalidate its own cache entry, and the client has supplied its own validator in the request, the supplied validator may differ from the validator currently stored with the cache entry. In this case, the cache may use either validator in making its own request without affecting semantic transparency.

However, the choice of validator may affect performance. The best approach is for the intermediate cache to use its own validator when making its request. If the server replies with 304 (Not Modified), then the cache should return its now validated copy to the client with a 200 (OK) response. If the server replies with a new entity and cache validator, however, the intermediate cache should compare the returned validator with the one provided in the client's request, using the strong comparison function. If the client's validator is equal to the origin server's, then the intermediate cache simply returns 304 (Not Modified). Otherwise, it returns the new entity with a 200 (OK) response.

If a request includes the no-cache directive, it should not include min-fresh, max-stale, or max-age.

In some cases, such as times of extremely poor network connectivity, a client may want a cache to return only those responses that it currently has stored, and not to reload or revalidate with the origin server. To do this, the client may include the only-if-cached directive in a request. If it receives this directive, a cache SHOULD either respond using a cached entry that is consistent with the other constraints of the request, or respond with a 504 (Gateway Timeout) status. However, if a group of caches is being operated as a unified system with good internal connectivity, such a request MAY be forwarded within that group of caches.

Because a cache may be configured to ignore a server's specified expiration time, and because a client request may include a max-stale directive (which has a similar effect), the protocol also includes a

mechanism for the origin server to require revalidation of a cache entry on any subsequent use. When the must-revalidate directive is present in a response received by a cache, that cache MUST NOT use the entry after it becomes stale to respond to a subsequent request without first revalidating it with the origin server. (I.e., the cache must do an end-to-end revalidation every time, if, based solely on the origin server's Expires or max-age value, the cached response is stale.)

The must-revalidate directive is necessary to support reliable operation for certain protocol features. In all circumstances an HTTP/1.1 cache MUST obey the must-revalidate directive; in particular, if the cache cannot reach the origin server for any reason, it MUST generate a 504 (Gateway Timeout) response.

Servers should send the must-revalidate directive if and only if failure to revalidate a request on the entity could result in incorrect operation, such as a silently unexecuted financial transaction. Recipients MUST NOT take any automated action that violates this directive, and MUST NOT automatically provide an unvalidated copy of the entity if revalidation fails.

Although this is not recommended, user agents operating under severe connectivity constraints may violate this directive but, if so, MUST explicitly warn the user that an unvalidated response has been provided. The warning MUST be provided on each unvalidated access, and SHOULD require explicit user confirmation.

The proxy-revalidate directive has the same meaning as the must-revalidate directive, except that it does not apply to non-shared user agent caches. It can be used on a response to an authenticated request to permit the user's cache to store and later return the response without needing to revalidate it (since it has already been authenticated once by that user), while still requiring proxies that service many users to revalidate each time (in order to make sure that each user has been authenticated). Note that such authenticated responses also need the public cache control directive in order to allow them to be cached at all.

14.9.5 No-Transform Directive

Implementers of intermediate caches (proxies) have found it useful to convert the media type of certain entity bodies. A proxy might, for example, convert between image formats in order to save cache space or to reduce the amount of traffic on a slow link. HTTP has to date been silent on these transformations.

Serious operational problems have already occurred, however, when these transformations have been applied to entity bodies intended for certain kinds of applications. For example, applications for medical imaging, scientific data analysis and those using end-to-end authentication, all depend on receiving an entity body that is bit for bit identical to the original entity-body.

Therefore, if a response includes the no-transform directive, an intermediate cache or proxy MUST NOT change those headers that are listed in section 13.5.2 as being subject to the no-transform directive. This implies that the cache or proxy must not change any aspect of the entity-body that is specified by these headers.

14.9.6 Cache Control Extensions

The Cache-Control header field can be extended through the use of one or more cache-extension tokens, each with an optional assigned value. Informational extensions (those which do not require a change in cache behavior) may be added without changing the semantics of other directives. Behavioral extensions are designed to work by acting as modifiers to the existing base of cache directives. Both the new directive and the standard directive are supplied, such that applications which do not understand the new directive will default to the behavior specified by the standard directive, and those that understand the new directive will recognize it as modifying the requirements associated with the standard directive. In this way, extensions to the Cache-Control directives can be made without requiring changes to the base protocol.

This extension mechanism depends on a HTTP cache obeying all of the cache-control directives defined for its native HTTP-version, obeying certain extensions, and ignoring all directives that it does not understand.

For example, consider a hypothetical new response directive called "community" which acts as a modifier to the "private" directive. We define this new directive to mean that, in addition to any non-shared cache, any cache which is shared only by members of the community named within its value may cache the response. An origin server wishing to allow the "UCI" community to use an otherwise private response in their shared cache(s) may do so by including

```
Cache-Control: private, community="UCI"
```

A cache seeing this header field will act correctly even if the cache does not understand the "community" cache-extension, since it will also see and understand the "private" directive and thus default to the safe behavior.

Unrecognized cache-directives MUST be ignored; it is assumed that any cache-directive likely to be unrecognized by an HTTP/1.1 cache will be combined with standard directives (or the response's default cachability) such that the cache behavior will remain minimally correct even if the cache does not understand the extension(s).

14.10 Connection

The Connection general-header field allows the sender to specify options that are desired for that particular connection and MUST NOT be communicated by proxies over further connections.

The Connection header has the following grammar:

```
Connection-header = "Connection" ":" 1#(connection-token)
connection-token  = token
```

HTTP/1.1 proxies MUST parse the Connection header field before a message is forwarded and, for each connection-token in this field, remove any header field(s) from the message with the same name as the connection-token. Connection options are signaled by the presence of a connection-token in the Connection header field, not by any corresponding additional header field(s), since the additional header field may not be sent if there are no parameters associated with that connection option. HTTP/1.1 defines the "close" connection option for the sender to signal that the connection will be closed after completion of the response. For example,

```
Connection: close
```

in either the request or the response header fields indicates that the connection should not be considered `persistent' (section 8.1) after the current request/response is complete.

HTTP/1.1 applications that do not support persistent connections MUST include the "close" connection option in every message.

14.11 Content-Base

The Content-Base entity-header field may be used to specify the base URI for resolving relative URLs within the entity. This header field is described as Base in RFC 1808, which is expected to be revised.

```
Content-Base      = "Content-Base" ":" absoluteURI
```

If no Content-Base field is present, the base URI of an entity is defined either by its Content-Location (if that Content-Location URI is an absolute URI) or the URI used to initiate the request, in that

order of precedence. Note, however, that the base URI of the contents within the entity-body may be redefined within that entity-body.

14.12 Content-Encoding

The Content-Encoding entity-header field is used as a modifier to the media-type. When present, its value indicates what additional content codings have been applied to the entity-body, and thus what decoding mechanisms MUST be applied in order to obtain the media-type referenced by the Content-Type header field. Content-Encoding is primarily used to allow a document to be compressed without losing the identity of its underlying media type.

```
Content-Encoding  = "Content-Encoding" ":" 1#content-coding
```

Content codings are defined in section 3.5. An example of its use is

```
Content-Encoding: gzip
```

The Content-Encoding is a characteristic of the entity identified by the Request-URI. Typically, the entity-body is stored with this encoding and is only decoded before rendering or analogous usage.

If multiple encodings have been applied to an entity, the content codings MUST be listed in the order in which they were applied.

Additional information about the encoding parameters MAY be provided by other entity-header fields not defined by this specification.

14.13 Content-Language

The Content-Language entity-header field describes the natural language(s) of the intended audience for the enclosed entity. Note that this may not be equivalent to all the languages used within the entity-body.

```
Content-Language  = "Content-Language" ":" 1#language-tag
```

Language tags are defined in section 3.10. The primary purpose of Content-Language is to allow a user to identify and differentiate entities according to the user's own preferred language. Thus, if the body content is intended only for a Danish-literate audience, the appropriate field is

```
Content-Language: da
```

If no Content-Language is specified, the default is that the content is intended for all language audiences. This may mean that the sender

does not consider it to be specific to any natural language, or that the sender does not know for which language it is intended.

Multiple languages MAY be listed for content that is intended for multiple audiences. For example, a rendition of the "Treaty of Waitangi," presented simultaneously in the original Maori and English versions, would call for

```
Content-Language: mi, en
```

However, just because multiple languages are present within an entity does not mean that it is intended for multiple linguistic audiences. An example would be a beginner's language primer, such as "A First Lesson in Latin," which is clearly intended to be used by an English-literate audience. In this case, the Content-Language should only include "en".

Content-Language may be applied to any media type -- it is not limited to textual documents.

14.14 Content-Length

The Content-Length entity-header field indicates the size of the message-body, in decimal number of octets, sent to the recipient or, in the case of the HEAD method, the size of the entity-body that would have been sent had the request been a GET.

```
Content-Length    = "Content-Length" ":" 1*DIGIT
```

An example is

```
Content-Length: 3495
```

Applications SHOULD use this field to indicate the size of the message-body to be transferred, regardless of the media type of the entity. It must be possible for the recipient to reliably determine the end of HTTP/1.1 requests containing an entity-body, e.g., because the request has a valid Content-Length field, uses Transfer-Encoding: chunked or a multipart body.

Any Content-Length greater than or equal to zero is a valid value. Section 4.4 describes how to determine the length of a message-body if a Content-Length is not given.

Note: The meaning of this field is significantly different from the corresponding definition in MIME, where it is an optional field used within the "message/external-body" content-type. In HTTP, it SHOULD be sent whenever the message's length can be determined prior to being transferred.

14.15 Content-Location

The Content-Location entity-header field may be used to supply the resource location for the entity enclosed in the message. In the case where a resource has multiple entities associated with it, and those entities actually have separate locations by which they might be individually accessed, the server should provide a Content-Location for the particular variant which is returned. In addition, a server SHOULD provide a Content-Location for the resource corresponding to the response entity.

```
Content-Location = "Content-Location" ":"
                   ( absoluteURI | relativeURI )
```

If no Content-Base header field is present, the value of Content-Location also defines the base URL for the entity (see section 14.11).

The Content-Location value is not a replacement for the original requested URI; it is only a statement of the location of the resource corresponding to this particular entity at the time of the request. Future requests MAY use the Content-Location URI if the desire is to identify the source of that particular entity.

A cache cannot assume that an entity with a Content-Location different from the URI used to retrieve it can be used to respond to later requests on that Content-Location URI. However, the Content-Location can be used to differentiate between multiple entities retrieved from a single requested resource, as described in section 13.6.

If the Content-Location is a relative URI, the URI is interpreted relative to any Content-Base URI provided in the response. If no Content-Base is provided, the relative URI is interpreted relative to the Request-URI.

14.16 Content-MD5

The Content-MD5 entity-header field, as defined in RFC 1864 [23], is an MD5 digest of the entity-body for the purpose of providing an end-to-end message integrity check (MIC) of the entity-body. (Note: a MIC is good for detecting accidental modification of the entity-body in transit, but is not proof against malicious attacks.)

```
Content-MD5   = "Content-MD5" ":" md5-digest

md5-digest   = <base64 of 128 bit MD5 digest as per RFC 1864>
```

The Content-MD5 header field may be generated by an origin server to function as an integrity check of the entity-body. Only origin servers may generate the Content-MD5 header field; proxies and gateways MUST NOT generate it, as this would defeat its value as an end-to-end integrity check. Any recipient of the entity-body, including gateways and proxies, MAY check that the digest value in this header field matches that of the entity-body as received.

The MD5 digest is computed based on the content of the entity-body, including any Content-Encoding that has been applied, but not including any Transfer-Encoding that may have been applied to the message-body. If the message is received with a Transfer-Encoding, that encoding must be removed prior to checking the Content-MD5 value against the received entity.

This has the result that the digest is computed on the octets of the entity-body exactly as, and in the order that, they would be sent if no Transfer-Encoding were being applied.

HTTP extends RFC 1864 to permit the digest to be computed for MIME composite media-types (e.g., multipart/* and message/rfc822), but this does not change how the digest is computed as defined in the preceding paragraph.

> Note: There are several consequences of this. The entity-body for composite types may contain many body-parts, each with its own MIME and HTTP headers (including Content-MD5, Content-Transfer-Encoding, and Content-Encoding headers). If a body-part has a Content-Transfer-Encoding or Content-Encoding header, it is assumed that the content of the body-part has had the encoding applied, and the body-part is included in the Content-MD5 digest as is -- i.e., after the application. The Transfer-Encoding header field is not allowed within body-parts.

> Note: while the definition of Content-MD5 is exactly the same for HTTP as in RFC 1864 for MIME entity-bodies, there are several ways in which the application of Content-MD5 to HTTP entity-bodies differs from its application to MIME entity-bodies. One is that HTTP, unlike MIME, does not use Content-Transfer-Encoding, and does use Transfer-Encoding and Content-Encoding. Another is that HTTP more frequently uses binary content types than MIME, so it is worth noting that, in such cases, the byte order used to compute the digest is the transmission byte order defined for the type. Lastly, HTTP allows transmission of text types with any of several line break conventions and not just the canonical form using CRLF. Conversion of all line breaks to CRLF should not be done before computing or checking the digest: the line break convention used in the text actually transmitted should be left unaltered when computing the digest.

14.17 Content-Range

The Content-Range entity-header is sent with a partial entity-body to specify where in the full entity-body the partial body should be inserted. It also indicates the total size of the full entity-body. When a server returns a partial response to a client, it must describe both the extent of the range covered by the response, and the length of the entire entity-body.

```
Content-Range = "Content-Range" ":" content-range-spec

content-range-spec      = byte-content-range-spec

byte-content-range-spec = bytes-unit SP first-byte-pos "-"
                          last-byte-pos "/" entity-length

entity-length           = 1*DIGIT
```

Unlike byte-ranges-specifier values, a byte-content-range-spec may only specify one range, and must contain absolute byte positions for both the first and last byte of the range.

A byte-content-range-spec whose last-byte-pos value is less than its first-byte-pos value, or whose entity-length value is less than or equal to its last-byte-pos value, is invalid. The recipient of an invalid byte-content-range-spec MUST ignore it and any content transferred along with it.

Examples of byte-content-range-spec values, assuming that the entity contains a total of 1234 bytes:

- The first 500 bytes:

  bytes 0-499/1234

- The second 500 bytes:

  bytes 500-999/1234

- All except for the first 500 bytes:

  bytes 500-1233/1234

- The last 500 bytes:

  bytes 734-1233/1234

When an HTTP message includes the content of a single range (for example, a response to a request for a single range, or to a request for a set of ranges that overlap without any holes), this content is transmitted with a Content-Range header, and a Content-Length header showing the number of bytes actually transferred. For example,

```
HTTP/1.1 206 Partial content
Date: Wed, 15 Nov 1995 06:25:24 GMT
Last-modified: Wed, 15 Nov 1995 04:58:08 GMT
Content-Range: bytes 21010-47021/47022
Content-Length: 26012
Content-Type: image/gif
```

When an HTTP message includes the content of multiple ranges (for example, a response to a request for multiple non-overlapping ranges), these are transmitted as a multipart MIME message. The multipart MIME content-type used for this purpose is defined in this specification to be "multipart/byteranges". See appendix 19.2 for its definition.

A client that cannot decode a MIME multipart/byteranges message should not ask for multiple byte-ranges in a single request.

When a client requests multiple byte-ranges in one request, the server SHOULD return them in the order that they appeared in the request.

If the server ignores a byte-range-spec because it is invalid, the server should treat the request as if the invalid Range header field

did not exist. (Normally, this means return a 200 response containing the full entity). The reason is that the only time a client will make such an invalid request is when the entity is smaller than the entity retrieved by a prior request.

14.18 Content-Type

The Content-Type entity-header field indicates the media type of the entity-body sent to the recipient or, in the case of the HEAD method, the media type that would have been sent had the request been a GET.

```
Content-Type   = "Content-Type" ":" media-type
```

Media types are defined in section 3.7. An example of the field is

```
Content-Type: text/html; charset=ISO-8859-4
```

Further discussion of methods for identifying the media type of an entity is provided in section 7.2.1.

14.19 Date

The Date general-header field represents the date and time at which the message was originated, having the same semantics as orig-date in RFC 822. The field value is an HTTP-date, as described in section 3.3.1.

```
Date  = "Date" ":" HTTP-date
```

An example is

```
Date: Tue, 15 Nov 1994 08:12:31 GMT
```

If a message is received via direct connection with the user agent (in the case of requests) or the origin server (in the case of responses), then the date can be assumed to be the current date at the receiving end. However, since the date--as it is believed by the origin--is important for evaluating cached responses, origin servers MUST include a Date header field in all responses. Clients SHOULD only send a Date header field in messages that include an entity-body, as in the case of the PUT and POST requests, and even then it is optional. A received message which does not have a Date header field SHOULD be assigned one by the recipient if the message will be cached by that recipient or gatewayed via a protocol which requires a Date.

In theory, the date SHOULD represent the moment just before the entity is generated. In practice, the date can be generated at any time during the message origination without affecting its semantic value.

The format of the Date is an absolute date and time as defined by HTTP-date in section 3.3; it MUST be sent in RFC1123 [8]-date format.

14.20 ETag

The ETag entity-header field defines the entity tag for the associated entity. The headers used with entity tags are described in sections 14.20, 14.25, 14.26 and 14.43. The entity tag may be used for comparison with other entities from the same resource (see section 13.3.2).

```
ETag = "ETag" ":" entity-tag
```

Examples:

```
ETag: "xyzzy"
ETag: W/"xyzzy"
ETag: ""
```

14.21 Expires

The Expires entity-header field gives the date/time after which the response should be considered stale. A stale cache entry may not normally be returned by a cache (either a proxy cache or an user agent cache) unless it is first validated with the origin server (or with an intermediate cache that has a fresh copy of the entity). See section 13.2 for further discussion of the expiration model.

The presence of an Expires field does not imply that the original resource will change or cease to exist at, before, or after that time.

The format is an absolute date and time as defined by HTTP-date in section 3.3; it MUST be in RFC1123-date format:

```
Expires = "Expires" ":" HTTP-date
```


An example of its use is

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
```

> Note: if a response includes a Cache-Control field with the max-age directive, that directive overrides the Expires field.

HTTP/1.1 clients and caches MUST treat other invalid date formats, especially including the value "0", as in the past (i.e., "already expired").

To mark a response as "already expired," an origin server should use an Expires date that is equal to the Date header value. (See the rules for expiration calculations in section 13.2.4.)

To mark a response as "never expires," an origin server should use an Expires date approximately one year from the time the response is sent. HTTP/1.1 servers should not send Expires dates more than one year in the future.

The presence of an Expires header field with a date value of some time in the future on an response that otherwise would by default be non-cacheable indicates that the response is cachable, unless indicated otherwise by a Cache-Control header field (section 14.9).

14.22 From

The From request-header field, if given, SHOULD contain an Internet e-mail address for the human user who controls the requesting user agent. The address SHOULD be machine-usable, as defined by mailbox in RFC 822 (as updated by RFC 1123):

```
From = "From" ":" mailbox
```

An example is:

```
From: webmaster@w3.org
```

This header field MAY be used for logging purposes and as a means for identifying the source of invalid or unwanted requests. It SHOULD NOT be used as an insecure form of access protection. The interpretation of this field is that the request is being performed on behalf of the person given, who accepts responsibility for the method performed. In particular, robot agents SHOULD include this header so that the person responsible for running the robot can be contacted if problems occur on the receiving end.

The Internet e-mail address in this field MAY be separate from the Internet host which issued the request. For example, when a request is passed through a proxy the original issuer's address SHOULD be used.

Note: The client SHOULD not send the From header field without the user's approval, as it may conflict with the user's privacy interests or their site's security policy. It is strongly recommended that the user be able to disable, enable, and modify the value of this field at any time prior to a request.

14.23 Host

The Host request-header field specifies the Internet host and port number of the resource being requested, as obtained from the original URL given by the user or referring resource (generally an HTTP URL, as described in section 3.2.2). The Host field value MUST represent the network location of the origin server or gateway given by the original URL. This allows the origin server or gateway to differentiate between internally-ambiguous URLs, such as the root "/" URL of a server for multiple host names on a single IP address.

```
Host = "Host" ":" host [ ":" port ]    ; Section 3.2.2
```

A "host" without any trailing port information implies the default port for the service requested (e.g., "80" for an HTTP URL). For example, a request on the origin server for <http://www.w3.org/pub/WWW/> MUST include:

```
GET /pub/WWW/ HTTP/1.1
Host: www.w3.org
```

A client MUST include a Host header field in all HTTP/1.1 request messages on the Internet (i.e., on any message corresponding to a request for a URL which includes an Internet host address for the service being requested). If the Host field is not already present, an HTTP/1.1 proxy MUST add a Host field to the request message prior to forwarding it on the Internet. All Internet-based HTTP/1.1 servers MUST respond with a 400 status code to any HTTP/1.1 request message which lacks a Host header field.

See sections 5.2 and 19.5.1 for other requirements relating to Host.

14.24 If-Modified-Since

The If-Modified-Since request-header field is used with the GET method to make it conditional: if the requested variant has not been modified since the time specified in this field, an entity will not

be returned from the server; instead, a 304 (not modified) response will be returned without any message-body.

```
If-Modified-Since = "If-Modified-Since" ":" HTTP-date
```

An example of the field is:

```
If-Modified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

A GET method with an If-Modified-Since header and no Range header requests that the identified entity be transferred only if it has been modified since the date given by the If-Modified-Since header. The algorithm for determining this includes the following cases:

a)If the request would normally result in anything other than a 200 (OK) status, or if the passed If-Modified-Since date is invalid, the response is exactly the same as for a normal GET. A date which is later than the server's current time is invalid.

b)If the variant has been modified since the If-Modified-Since date, the response is exactly the same as for a normal GET.

c)If the variant has not been modified since a valid If-Modified-Since date, the server MUST return a 304 (Not Modified) response.

The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead.

Note that the Range request-header field modifies the meaning of If-Modified-Since; see section 14.36 for full details.

Note that If-Modified-Since times are interpreted by the server, whose clock may not be synchronized with the client.

Note that if a client uses an arbitrary date in the If-Modified-Since header instead of a date taken from the Last-Modified header for the same request, the client should be aware of the fact that this date is interpreted in the server's understanding of time. The client should consider unsynchronized clocks and rounding problems due to the different encodings of time between the client and server. This includes the possibility of race conditions if the document has changed between the time it was first requested and the If-Modified-Since date of a subsequent request, and the possibility of clock-skew-related problems if the If-Modified-Since date is derived from the client's clock without correction to the server's clock. Corrections for different time bases between client and server are at best approximate due to network latency.

14.25 If-Match

The If-Match request-header field is used with a method to make it conditional. A client that has one or more entities previously obtained from the resource can verify that one of those entities is current by including a list of their associated entity tags in the If-Match header field. The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead. It is also used, on updating requests, to prevent inadvertent modification of the wrong version of a resource. As a special case, the value "*" matches any current entity of the resource.

```
If-Match = "If-Match" ":" ( "*" | 1#entity-tag )
```

If any of the entity tags match the entity tag of the entity that would have been returned in the response to a similar GET request (without the If-Match header) on that resource, or if "*" is given and any current entity exists for that resource, then the server MAY perform the requested method as if the If-Match header field did not exist.

A server MUST use the strong comparison function (see section 3.11) to compare the entity tags in If-Match.

If none of the entity tags match, or if "*" is given and no current entity exists, the server MUST NOT perform the requested method, and MUST return a 412 (Precondition Failed) response. This behavior is most useful when the client wants to prevent an updating method, such as PUT, from modifying a resource that has changed since the client last retrieved it.

If the request would, without the If-Match header field, result in anything other than a 2xx status, then the If-Match header MUST be ignored.

The meaning of "If-Match: *" is that the method SHOULD be performed if the representation selected by the origin server (or by a cache, possibly using the Vary mechanism, see section 14.43) exists, and MUST NOT be performed if the representation does not exist.

A request intended to update a resource (e.g., a PUT) MAY include an If-Match header field to signal that the request method MUST NOT be applied if the entity corresponding to the If-Match value (a single entity tag) is no longer a representation of that resource. This allows the user to indicate that they do not wish the request to be successful if the resource has been changed without their knowledge. Examples:

```
If-Match: "xyzzy"
If-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-Match: *
```

14.26 If-None-Match

The If-None-Match request-header field is used with a method to make it conditional. A client that has one or more entities previously obtained from the resource can verify that none of those entities is current by including a list of their associated entity tags in the If-None-Match header field. The purpose of this feature is to allow efficient updates of cached information with a minimum amount of transaction overhead. It is also used, on updating requests, to prevent inadvertent modification of a resource which was not known to exist.

As a special case, the value "*" matches any current entity of the resource.

```
If-None-Match = "If-None-Match" ":" ( "*" | 1#entity-tag )
```

If any of the entity tags match the entity tag of the entity that would have been returned in the response to a similar GET request (without the If-None-Match header) on that resource, or if "*" is given and any current entity exists for that resource, then the server MUST NOT perform the requested method. Instead, if the request method was GET or HEAD, the server SHOULD respond with a 304 (Not Modified) response, including the cache-related entity-header fields (particularly ETag) of one of the entities that matched. For all other request methods, the server MUST respond with a status of 412 (Precondition Failed).

See section 13.3.3 for rules on how to determine if two entity tags match. The weak comparison function can only be used with GET or HEAD requests.

If none of the entity tags match, or if "*" is given and no current entity exists, then the server MAY perform the requested method as if the If-None-Match header field did not exist.

If the request would, without the If-None-Match header field, result in anything other than a 2xx status, then the If-None-Match header MUST be ignored.

The meaning of "If-None-Match: *" is that the method MUST NOT be performed if the representation selected by the origin server (or by a cache, possibly using the Vary mechanism, see section 14.43) exists, and SHOULD be performed if the representation does not exist. This feature may be useful in preventing races between PUT operations.

Examples:

```
If-None-Match: "xyzzy"
If-None-Match: W/"xyzzy"
If-None-Match: "xyzzy", "r2d2xxxx", "c3piozzzz"
If-None-Match: W/"xyzzy", W/"r2d2xxxx", W/"c3piozzzz"
If-None-Match: *
```

14.27 If-Range

If a client has a partial copy of an entity in its cache, and wishes to have an up-to-date copy of the entire entity in its cache, it could use the Range request-header with a conditional GET (using either or both of If-Unmodified-Since and If-Match.) However, if the condition fails because the entity has been modified, the client would then have to make a second request to obtain the entire current entity-body.

The If-Range header allows a client to "short-circuit" the second request. Informally, its meaning is `if the entity is unchanged, send me the part(s) that I am missing; otherwise, send me the entire new entity.'

```
If-Range = "If-Range" ":" ( entity-tag | HTTP-date )
```

If the client has no entity tag for an entity, but does have a Last-Modified date, it may use that date in a If-Range header. (The server can distinguish between a valid HTTP-date and any form of entity-tag by examining no more than two characters.) The If-Range header should only be used together with a Range header, and must be ignored if the request does not include a Range header, or if the server does not support the sub-range operation.

If the entity tag given in the If-Range header matches the current entity tag for the entity, then the server should provide the specified sub-range of the entity using a 206 (Partial content) response. If the entity tag does not match, then the server should return the entire entity using a 200 (OK) response.

14.28 If-Unmodified-Since

The If-Unmodified-Since request-header field is used with a method to make it conditional. If the requested resource has not been modified since the time specified in this field, the server should perform the requested operation as if the If-Unmodified-Since header were not present.

If the requested variant has been modified since the specified time, the server MUST NOT perform the requested operation, and MUST return a 412 (Precondition Failed).

```
If-Unmodified-Since = "If-Unmodified-Since" ":" HTTP-date
```

An example of the field is:

```
If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

If the request normally (i.e., without the If-Unmodified-Since header) would result in anything other than a 2xx status, the If-Unmodified-Since header should be ignored.

If the specified date is invalid, the header is ignored.

14.29 Last-Modified

The Last-Modified entity-header field indicates the date and time at which the origin server believes the variant was last modified.

```
Last-Modified  = "Last-Modified" ":" HTTP-date
```

An example of its use is

```
Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
```

The exact meaning of this header field depends on the implementation of the origin server and the nature of the original resource. For files, it may be just the file system last-modified time. For entities with dynamically included parts, it may be the most recent of the set of last-modify times for its component parts. For database gateways, it may be the last-update time stamp of the record. For virtual objects, it may be the last time the internal state changed.

An origin server MUST NOT send a Last-Modified date which is later than the server's time of message origination. In such cases, where the resource's last modification would indicate some time in the future, the server MUST replace that date with the message origination date.

An origin server should obtain the Last-Modified value of the entity as close as possible to the time that it generates the Date value of its response. This allows a recipient to make an accurate assessment of the entity's modification time, especially if the entity changes near the time that the response is generated.

HTTP/1.1 servers SHOULD send Last-Modified whenever feasible.

14.30 Location

The Location response-header field is used to redirect the recipient to a location other than the Request-URI for completion of the request or identification of a new resource. For 201 (Created) responses, the Location is that of the new resource which was created by the request. For 3xx responses, the location SHOULD indicate the server's preferred URL for automatic redirection to the resource. The field value consists of a single absolute URL.

```
Location       = "Location" ":" absoluteURI
```

An example is

```
Location: http://www.w3.org/pub/WWW/People.html
```

Note: The Content-Location header field (section 14.15) differs from Location in that the Content-Location identifies the original location of the entity enclosed in the request. It is therefore possible for a response to contain header fields for both Location and Content-Location. Also see section 13.10 for cache requirements of some methods.

14.31 Max-Forwards

The Max-Forwards request-header field may be used with the TRACE method (section 14.31) to limit the number of proxies or gateways that can forward the request to the next inbound server. This can be useful when the client is attempting to trace a request chain which appears to be failing or looping in mid-chain.

```
Max-Forwards   = "Max-Forwards" ":" 1*DIGIT
```


The Max-Forwards value is a decimal integer indicating the remaining number of times this request message may be forwarded.

Each proxy or gateway recipient of a TRACE request containing a Max-Forwards header field SHOULD check and update its value prior to forwarding the request. If the received value is zero (0), the recipient SHOULD NOT forward the request; instead, it SHOULD respond as the final recipient with a 200 (OK) response containing the received request message as the response entity-body (as described in section 9.8). If the received Max-Forwards value is greater than zero, then the forwarded message SHOULD contain an updated Max-Forwards field with a value decremented by one (1).

The Max-Forwards header field SHOULD be ignored for all other methods defined by this specification and for any extension methods for which it is not explicitly referred to as part of that method definition.

14.32 Pragma

The Pragma general-header field is used to include implementation-specific directives that may apply to any recipient along the request/response chain. All pragma directives specify optional behavior from the viewpoint of the protocol; however, some systems MAY require that behavior be consistent with the directives.

```
Pragma            = "Pragma" ":" 1#pragma-directive

pragma-directive  = "no-cache" | extension-pragma
extension-pragma  = token [ "=" ( token | quoted-string ) ]
```

When the no-cache directive is present in a request message, an application SHOULD forward the request toward the origin server even if it has a cached copy of what is being requested. This pragma directive has the same semantics as the no-cache cache-directive (see section 14.9) and is defined here for backwards compatibility with HTTP/1.0. Clients SHOULD include both header fields when a no-cache request is sent to a server not known to be HTTP/1.1 compliant.

Pragma directives MUST be passed through by a proxy or gateway application, regardless of their significance to that application, since the directives may be applicable to all recipients along the request/response chain. It is not possible to specify a pragma for a specific recipient; however, any pragma directive not relevant to a recipient SHOULD be ignored by that recipient.

HTTP/1.1 clients SHOULD NOT send the Pragma request-header. HTTP/1.1 caches SHOULD treat "Pragma: no-cache" as if the client had sent "Cache-Control: no-cache". No new Pragma directives will be defined in HTTP.

14.33 Proxy-Authenticate

The Proxy-Authenticate response-header field MUST be included as part of a 407 (Proxy Authentication Required) response. The field value consists of a challenge that indicates the authentication scheme and parameters applicable to the proxy for this Request-URI.

```
Proxy-Authenticate  = "Proxy-Authenticate" ":" challenge
```

The HTTP access authentication process is described in section 11. Unlike WWW-Authenticate, the Proxy-Authenticate header field applies only to the current connection and SHOULD NOT be passed on to downstream clients. However, an intermediate proxy may need to obtain its own credentials by requesting them from the downstream client, which in some circumstances will appear as if the proxy is forwarding the Proxy-Authenticate header field.

14.34 Proxy-Authorization

The Proxy-Authorization request-header field allows the client to identify itself (or its user) to a proxy which requires authentication. The Proxy-Authorization field value consists of credentials containing the authentication information of the user agent for the proxy and/or realm of the resource being requested.

```
Proxy-Authorization     = "Proxy-Authorization" ":" credentials
```

The HTTP access authentication process is described in section 11. Unlike Authorization, the Proxy-Authorization header field applies only to the next outbound proxy that demanded authentication using the Proxy-Authenticate field. When multiple proxies are used in a chain, the Proxy-Authorization header field is consumed by the first outbound proxy that was expecting to receive credentials. A proxy MAY relay the credentials from the client request to the next proxy if that is the mechanism by which the proxies cooperatively authenticate a given request.

14.35 Public

The Public response-header field lists the set of methods supported by the server. The purpose of this field is strictly to inform the recipient of the capabilities of the server regarding unusual methods. The methods listed may or may not be applicable to the

Request-URI; the Allow header field (section 14.7) MAY be used to indicate methods allowed for a particular URI.

```
Public         = "Public" ":" 1#method
```

Example of use:

```
Public: OPTIONS, MGET, MHEAD, GET, HEAD
```

This header field applies only to the server directly connected to the client (i.e., the nearest neighbor in a chain of connections). If the response passes through a proxy, the proxy MUST either remove the Public header field or replace it with one applicable to its own capabilities.

14.36 Range

14.36.1 Byte Ranges

Since all HTTP entities are represented in HTTP messages as sequences of bytes, the concept of a byte range is meaningful for any HTTP entity. (However, not all clients and servers need to support byte-range operations.)

Byte range specifications in HTTP apply to the sequence of bytes in the entity-body (not necessarily the same as the message-body).

A byte range operation may specify a single range of bytes, or a set of ranges within a single entity.

```
ranges-specifier = byte-ranges-specifier

byte-ranges-specifier = bytes-unit "=" byte-range-set

byte-range-set  = 1#( byte-range-spec | suffix-byte-range-spec )

byte-range-spec = first-byte-pos "-" [last-byte-pos]

first-byte-pos  = 1*DIGIT

last-byte-pos   = 1*DIGIT
```

The first-byte-pos value in a byte-range-spec gives the byte-offset of the first byte in a range. The last-byte-pos value gives the byte-offset of the last byte in the range; that is, the byte positions specified are inclusive. Byte offsets start at zero.

If the last-byte-pos value is present, it must be greater than or equal to the first-byte-pos in that byte-range-spec, or the byte-range-spec is invalid. The recipient of an invalid byte-range-spec must ignore it.

If the last-byte-pos value is absent, or if the value is greater than or equal to the current length of the entity-body, last-byte-pos is taken to be equal to one less than the current length of the entity-body in bytes.

By its choice of last-byte-pos, a client can limit the number of bytes retrieved without knowing the size of the entity.

```
suffix-byte-range-spec = "-" suffix-length

suffix-length = 1*DIGIT
```

A suffix-byte-range-spec is used to specify the suffix of the entity-body, of a length given by the suffix-length value. (That is, this form specifies the last N bytes of an entity-body.) If the entity is shorter than the specified suffix-length, the entire entity-body is used.

Examples of byte-ranges-specifier values (assuming an entity-body of length 10000):

- The first 500 bytes (byte offsets 0-499, inclusive):

  ```
  bytes=0-499
  ```

- The second 500 bytes (byte offsets 500-999, inclusive):

  ```
  bytes=500-999
  ```

- The final 500 bytes (byte offsets 9500-9999, inclusive):

  ```
  bytes=-500
  ```

- Or

  ```
  bytes=9500-
  ```

- The first and last bytes only (bytes 0 and 9999):

  ```
  bytes=0-0,-1
  ```


- Several legal but not canonical specifications of the second 500 bytes (byte offsets 500-999, inclusive):

  ```
  bytes=500-600,601-999

  bytes=500-700,601-999
  ```

14.36.2 Range Retrieval Requests

HTTP retrieval requests using conditional or unconditional GET methods may request one or more sub-ranges of the entity, instead of the entire entity, using the Range request header, which applies to the entity returned as the result of the request:

```
Range = "Range" ":" ranges-specifier
```

A server MAY ignore the Range header. However, HTTP/1.1 origin servers and intermediate caches SHOULD support byte ranges when possible, since Range supports efficient recovery from partially failed transfers, and supports efficient partial retrieval of large entities.

If the server supports the Range header and the specified range or ranges are appropriate for the entity:

- The presence of a Range header in an unconditional GET modifies what is returned if the GET is otherwise successful. In other words, the response carries a status code of 206 (Partial Content) instead of 200 (OK).

- The presence of a Range header in a conditional GET (a request using one or both of If-Modified-Since and If-None-Match, or one or both of If-Unmodified-Since and If-Match) modifies what is returned if the GET is otherwise successful and the condition is true. It does not affect the 304 (Not Modified) response returned if the conditional is false.

In some cases, it may be more appropriate to use the If-Range header (see section 14.27) in addition to the Range header.

If a proxy that supports ranges receives a Range request, forwards the request to an inbound server, and receives an entire entity in reply, it SHOULD only return the requested range to its client. It SHOULD store the entire received response in its cache, if that is consistent with its cache allocation policies.

14.37 Referer

The Referer[sic] request-header field allows the client to specify, for the server's benefit, the address (URI) of the resource from which the Request-URI was obtained (the "referrer", although the header field is misspelled.) The Referer request-header allows a server to generate lists of back-links to resources for interest, logging, optimized caching, etc. It also allows obsolete or mistyped links to be traced for maintenance. The Referer field MUST NOT be sent if the Request-URI was obtained from a source that does not have its own URI, such as input from the user keyboard.

```
Referer        = "Referer" ":" ( absoluteURI | relativeURI )
```

Example:

```
Referer: http://www.w3.org/hypertext/DataSources/Overview.html
```

If the field value is a partial URI, it SHOULD be interpreted relative to the Request-URI. The URI MUST NOT include a fragment.

> Note: Because the source of a link may be private information or may reveal an otherwise private information source, it is strongly recommended that the user be able to select whether or not the Referer field is sent. For example, a browser client could have a toggle switch for browsing openly/anonymously, which would respectively enable/disable the sending of Referer and From information.

14.38 Retry-After

The Retry-After response-header field can be used with a 503 (Service Unavailable) response to indicate how long the service is expected to be unavailable to the requesting client. The value of this field can be either an HTTP-date or an integer number of seconds (in decimal) after the time of the response.

```
Retry-After  = "Retry-After" ":" ( HTTP-date | delta-seconds )
```

Two examples of its use are

```
Retry-After: Fri, 31 Dec 1999 23:59:59 GMT
Retry-After: 120
```

In the latter example, the delay is 2 minutes.

14.39 Server

The Server response-header field contains information about the software used by the origin server to handle the request. The field can contain multiple product tokens (section 3.8) and comments identifying the server and any significant subproducts. The product tokens are listed in order of their significance for identifying the application.

```
Server         = "Server" ":" 1*( product | comment )
```

Example:

```
Server: CERN/3.0 libwww/2.17
```

If the response is being forwarded through a proxy, the proxy application MUST NOT modify the Server response-header. Instead, it SHOULD include a Via field (as described in section 14.44).

> Note: Revealing the specific software version of the server may allow the server machine to become more vulnerable to attacks against software that is known to contain security holes. Server implementers are encouraged to make this field a configurable option.

14.40 Transfer-Encoding

The Transfer-Encoding general-header field indicates what (if any) type of transformation has been applied to the message body in order to safely transfer it between the sender and the recipient. This differs from the Content-Encoding in that the transfer coding is a property of the message, not of the entity.

```
Transfer-Encoding       = "Transfer-Encoding" ":" 1#transfer-coding
```

Transfer codings are defined in section 3.6. An example is:

```
Transfer-Encoding: chunked
```

Many older HTTP/1.0 applications do not understand the Transfer-Encoding header.

14.41 Upgrade

The Upgrade general-header allows the client to specify what additional communication protocols it supports and would like to use if the server finds it appropriate to switch protocols. The server

MUST use the Upgrade header field within a 101 (Switching Protocols) response to indicate which protocol(s) are being switched.

```
Upgrade        = "Upgrade" ":" 1#product
```

For example,

```
Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
```

The Upgrade header field is intended to provide a simple mechanism for transition from HTTP/1.1 to some other, incompatible protocol. It does so by allowing the client to advertise its desire to use another protocol, such as a later version of HTTP with a higher major version number, even though the current request has been made using HTTP/1.1. This eases the difficult transition between incompatible protocols by allowing the client to initiate a request in the more commonly supported protocol while indicating to the server that it would like to use a "better" protocol if available (where "better" is determined by the server, possibly according to the nature of the method and/or resource being requested).

The Upgrade header field only applies to switching application-layer protocols upon the existing transport-layer connection. Upgrade cannot be used to insist on a protocol change; its acceptance and use by the server is optional. The capabilities and nature of the application-layer communication after the protocol change is entirely dependent upon the new protocol chosen, although the first action after changing the protocol MUST be a response to the initial HTTP request containing the Upgrade header field.

The Upgrade header field only applies to the immediate connection. Therefore, the upgrade keyword MUST be supplied within a Connection header field (section 14.10) whenever Upgrade is present in an HTTP/1.1 message.

The Upgrade header field cannot be used to indicate a switch to a protocol on a different connection. For that purpose, it is more appropriate to use a 301, 302, 303, or 305 redirection response.

This specification only defines the protocol name "HTTP" for use by the family of Hypertext Transfer Protocols, as defined by the HTTP version rules of section 3.1 and future updates to this specification. Any token can be used as a protocol name; however, it will only be useful if both the client and server associate the name with the same protocol.

14.42 User-Agent

The User-Agent request-header field contains information about the user agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations. User agents SHOULD include this field with requests. The field can contain multiple product tokens (section 3.8) and comments identifying the agent and any subproducts which form a significant part of the user agent. By convention, the product tokens are listed in order of their significance for identifying the application.

```
User-Agent     = "User-Agent" ":" 1*( product | comment )
```

Example:

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
```

14.43 Vary

The Vary response-header field is used by a server to signal that the response entity was selected from the available representations of the response using server-driven negotiation (section 12). Field-names listed in Vary headers are those of request-headers. The Vary field value indicates either that the given set of header fields encompass the dimensions over which the representation might vary, or that the dimensions of variance are unspecified ("*") and thus may vary over any aspect of future requests.

```
Vary  = "Vary" ":" ( "*" | 1#field-name )
```

An HTTP/1.1 server MUST include an appropriate Vary header field with any cachable response that is subject to server-driven negotiation. Doing so allows a cache to properly interpret future requests on that resource and informs the user agent about the presence of negotiation on that resource. A server SHOULD include an appropriate Vary header field with a non-cachable response that is subject to server-driven negotiation, since this might provide the user agent with useful information about the dimensions over which the response might vary.

The set of header fields named by the Vary field value is known as the "selecting" request-headers.

When the cache receives a subsequent request whose Request-URI specifies one or more cache entries including a Vary header, the cache MUST NOT use such a cache entry to construct a response to the new request unless all of the headers named in the cached Vary header

are present in the new request, and all of the stored selecting request-headers from the previous request match the corresponding headers in the new request.

The selecting request-headers from two requests are defined to match if and only if the selecting request-headers in the first request can be transformed to the selecting request-headers in the second request by adding or removing linear whitespace (LWS) at places where this is allowed by the corresponding BNF, and/or combining multiple message-header fields with the same field name following the rules about message headers in section 4.2.

A Vary field value of "*" signals that unspecified parameters, possibly other than the contents of request-header fields (e.g., the network address of the client), play a role in the selection of the response representation. Subsequent requests on that resource can only be properly interpreted by the origin server, and thus a cache MUST forward a (possibly conditional) request even when it has a fresh response cached for the resource. See section 13.6 for use of the Vary header by caches.

A Vary field value consisting of a list of field-names signals that the representation selected for the response is based on a selection algorithm which considers ONLY the listed request-header field values in selecting the most appropriate representation. A cache MAY assume that the same selection will be made for future requests with the same values for the listed field names, for the duration of time in which the response is fresh.

The field-names given are not limited to the set of standard request-header fields defined by this specification. Field names are case-insensitive.

14.44 Via

The Via general-header field MUST be used by gateways and proxies to indicate the intermediate protocols and recipients between the user agent and the server on requests, and between the origin server and the client on responses. It is analogous to the "Received" field of RFC 822 and is intended to be used for tracking message forwards, avoiding request loops, and identifying the protocol capabilities of all senders along the request/response chain.


```
Via =  "Via" ":" 1#( received-protocol received-by [ comment ] )

received-protocol = [ protocol-name "/" ] protocol-version
protocol-name     = token
protocol-version  = token
received-by       = ( host [ ":" port ] ) | pseudonym
pseudonym         = token
```

The received-protocol indicates the protocol version of the message received by the server or client along each segment of the request/response chain. The received-protocol version is appended to the Via field value when the message is forwarded so that information about the protocol capabilities of upstream applications remains visible to all recipients.

The protocol-name is optional if and only if it would be "HTTP". The received-by field is normally the host and optional port number of a recipient server or client that subsequently forwarded the message. However, if the real host is considered to be sensitive information, it MAY be replaced by a pseudonym. If the port is not given, it MAY be assumed to be the default port of the received-protocol.

Multiple Via field values represent each proxy or gateway that has forwarded the message. Each recipient MUST append its information such that the end result is ordered according to the sequence of forwarding applications.

Comments MAY be used in the Via header field to identify the software of the recipient proxy or gateway, analogous to the User-Agent and Server header fields. However, all comments in the Via field are optional and MAY be removed by any recipient prior to forwarding the message.

For example, a request message could be sent from an HTTP/1.0 user agent to an internal proxy code-named "fred", which uses HTTP/1.1 to forward the request to a public proxy at nowhere.com, which completes the request by forwarding it to the origin server at www.ics.uci.edu. The request received by www.ics.uci.edu would then have the following Via header field:

```
Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)
```

Proxies and gateways used as a portal through a network firewall SHOULD NOT, by default, forward the names and ports of hosts within the firewall region. This information SHOULD only be propagated if explicitly enabled. If not enabled, the received-by host of any host behind the firewall SHOULD be replaced by an appropriate pseudonym for that host.

For organizations that have strong privacy requirements for hiding internal structures, a proxy MAY combine an ordered subsequence of Via header field entries with identical received-protocol values into a single such entry. For example,

```
Via: 1.0 ricky, 1.1 ethel, 1.1 fred, 1.0 lucy

 could be collapsed to

Via: 1.0 ricky, 1.1 mertz, 1.0 lucy
```

Applications SHOULD NOT combine multiple entries unless they are all under the same organizational control and the hosts have already been replaced by pseudonyms. Applications MUST NOT combine entries which have different received-protocol values.

14.45 Warning

The Warning response-header field is used to carry additional information about the status of a response which may not be reflected by the response status code. This information is typically, though not exclusively, used to warn about a possible lack of semantic transparency from caching operations.

Warning headers are sent with responses using:

```
Warning    = "Warning" ":" 1#warning-value

warning-value = warn-code SP warn-agent SP warn-text
warn-code  = 2DIGIT
warn-agent = ( host [ ":" port ] ) | pseudonym
                ; the name or pseudonym of the server adding
                ; the Warning header, for use in debugging
warn-text  = quoted-string
```

A response may carry more than one Warning header.

The warn-text should be in a natural language and character set that is most likely to be intelligible to the human user receiving the response. This decision may be based on any available knowledge, such as the location of the cache or user, the Accept-Language field in a request, the Content-Language field in a response, etc. The default language is English and the default character set is ISO-8859-1.

If a character set other than ISO-8859-1 is used, it MUST be encoded in the warn-text using the method described in RFC 1522 [14].

Any server or cache may add Warning headers to a response. New Warning headers should be added after any existing Warning headers. A cache MUST NOT delete any Warning header that it received with a response. However, if a cache successfully validates a cache entry, it SHOULD remove any Warning headers previously attached to that entry except as specified for specific Warning codes. It MUST then add any Warning headers received in the validating response. In other words, Warning headers are those that would be attached to the most recent relevant response.

When multiple Warning headers are attached to a response, the user agent SHOULD display as many of them as possible, in the order that they appear in the response. If it is not possible to display all of the warnings, the user agent should follow these heuristics:

- Warnings that appear early in the response take priority over those appearing later in the response.
- Warnings in the user's preferred character set take priority over warnings in other character sets but with identical warn-codes and warn-agents.

Systems that generate multiple Warning headers should order them with this user agent behavior in mind.

This is a list of the currently-defined warn-codes, each with a recommended warn-text in English, and a description of its meaning.

10 Response is stale
MUST be included whenever the returned response is stale. A cache may add this warning to any response, but may never remove it until the response is known to be fresh.

11 Revalidation failed
MUST be included if a cache returns a stale response because an attempt to revalidate the response failed, due to an inability to reach the server. A cache may add this warning to any response, but may never remove it until the response is successfully revalidated.

12 Disconnected operation
SHOULD be included if the cache is intentionally disconnected from the rest of the network for a period of time.

13 Heuristic expiration
MUST be included if the cache heuristically chose a freshness lifetime greater than 24 hours and the response's age is greater than 24 hours.

14 Transformation applied
MUST be added by an intermediate cache or proxy if it applies any transformation changing the content-coding (as specified in the Content-Encoding header) or media-type (as specified in the Content-Type header) of the response, unless this Warning code already appears in the response. MUST NOT be deleted from a response even after revalidation.

99 Miscellaneous warning
The warning text may include arbitrary information to be presented to a human user, or logged. A system receiving this warning MUST NOT take any automated action.

14.46 WWW-Authenticate

The WWW-Authenticate response-header field MUST be included in 401 (Unauthorized) response messages. The field value consists of at least one challenge that indicates the authentication scheme(s) and parameters applicable to the Request-URI.

```
WWW-Authenticate  = "WWW-Authenticate" ":" 1#challenge
```

The HTTP access authentication process is described in section 11. User agents MUST take special care in parsing the WWW-Authenticate field value if it contains more than one challenge, or if more than one WWW-Authenticate header field is provided, since the contents of a challenge may itself contain a comma-separated list of authentication parameters.

15 Security Considerations

This section is meant to inform application developers, information providers, and users of the security limitations in HTTP/1.1 as described by this document. The discussion does not include definitive solutions to the problems revealed, though it does make some suggestions for reducing security risks.

15.1 Authentication of Clients

The Basic authentication scheme is not a secure method of user authentication, nor does it in any way protect the entity, which is transmitted in clear text across the physical network used as the carrier. HTTP does not prevent additional authentication schemes and encryption mechanisms from being employed to increase security or the addition of enhancements (such as schemes to use one-time passwords) to Basic authentication.

The most serious flaw in Basic authentication is that it results in the essentially clear text transmission of the user's password over the physical network. It is this problem which Digest Authentication attempts to address.

Because Basic authentication involves the clear text transmission of passwords it SHOULD never be used (without enhancements) to protect sensitive or valuable information.

A common use of Basic authentication is for identification purposes -- requiring the user to provide a user name and password as a means of identification, for example, for purposes of gathering accurate usage statistics on a server. When used in this way it is tempting to think that there is no danger in its use if illicit access to the protected documents is not a major concern. This is only correct if the server issues both user name and password to the users and in particular does not allow the user to choose his or her own password. The danger arises because naive users frequently reuse a single password to avoid the task of maintaining multiple passwords.

If a server permits users to select their own passwords, then the threat is not only illicit access to documents on the server but also illicit access to the accounts of all users who have chosen to use their account password. If users are allowed to choose their own password that also means the server must maintain files containing the (presumably encrypted) passwords. Many of these may be the account passwords of users perhaps at distant sites. The owner or administrator of such a system could conceivably incur liability if this information is not maintained in a secure fashion.

Basic Authentication is also vulnerable to spoofing by counterfeit servers. If a user can be led to believe that he is connecting to a host containing information protected by basic authentication when in fact he is connecting to a hostile server or gateway then the attacker can request a password, store it for later use, and feign an error. This type of attack is not possible with Digest Authentication [32]. Server implementers SHOULD guard against the possibility of this sort of counterfeiting by gateways or CGI scripts. In particular it is very dangerous for a server to simply turn over a connection to a gateway since that gateway can then use the persistent connection mechanism to engage in multiple transactions with the client while impersonating the original server in a way that is not detectable by the client.

15.2 Offering a Choice of Authentication Schemes

An HTTP/1.1 server may return multiple challenges with a 401 (Authenticate) response, and each challenge may use a different

scheme. The order of the challenges returned to the user agent is in the order that the server would prefer they be chosen. The server should order its challenges with the "most secure" authentication scheme first. A user agent should choose as the challenge to be made to the user the first one that the user agent understands.

When the server offers choices of authentication schemes using the WWW-Authenticate header, the "security" of the authentication is only as malicious user could capture the set of challenges and try to authenticate him/herself using the weakest of the authentication schemes. Thus, the ordering serves more to protect the user's credentials than the server's information.

A possible man-in-the-middle (MITM) attack would be to add a weak authentication scheme to the set of choices, hoping that the client will use one that exposes the user's credentials (e.g. password). For this reason, the client should always use the strongest scheme that it understands from the choices accepted.

An even better MITM attack would be to remove all offered choices, and to insert a challenge that requests Basic authentication. For this reason, user agents that are concerned about this kind of attack could remember the strongest authentication scheme ever requested by a server and produce a warning message that requires user confirmation before using a weaker one. A particularly insidious way to mount such a MITM attack would be to offer a "free" proxy caching service to gullible users.

15.3 Abuse of Server Log Information

A server is in the position to save personal data about a user's requests which may identify their reading patterns or subjects of interest. This information is clearly confidential in nature and its handling may be constrained by law in certain countries. People using the HTTP protocol to provide data are responsible for ensuring that such material is not distributed without the permission of any individuals that are identifiable by the published results.

15.4 Transfer of Sensitive Information

Like any generic data transfer protocol, HTTP cannot regulate the content of the data that is transferred, nor is there any a priori method of determining the sensitivity of any particular piece of information within the context of any given request. Therefore, applications SHOULD supply as much control over this information as possible to the provider of that information. Four header fields are worth special mention in this context: Server, Via, Referer and From.

Revealing the specific software version of the server may allow the server machine to become more vulnerable to attacks against software that is known to contain security holes. Implementers SHOULD make the Server header field a configurable option.

Proxies which serve as a portal through a network firewall SHOULD take special precautions regarding the transfer of header information that identifies the hosts behind the firewall. In particular, they SHOULD remove, or replace with sanitized versions, any Via fields generated behind the firewall.

The Referer field allows reading patterns to be studied and reverse links drawn. Although it can be very useful, its power can be abused if user details are not separated from the information contained in the Referer. Even when the personal information has been removed, the Referer field may indicate a private document's URI whose publication would be inappropriate.

The information sent in the From field might conflict with the user's privacy interests or their site's security policy, and hence it SHOULD NOT be transmitted without the user being able to disable, enable, and modify the contents of the field. The user MUST be able to set the contents of this field within a user preference or application defaults configuration.

We suggest, though do not require, that a convenient toggle interface be provided for the user to enable or disable the sending of From and Referer information.

15.5 Attacks Based On File and Path Names

Implementations of HTTP origin servers SHOULD be careful to restrict the documents returned by HTTP requests to be only those that were intended by the server administrators. If an HTTP server translates HTTP URIs directly into file system calls, the server MUST take special care not to serve files that were not intended to be delivered to HTTP clients. For example, UNIX, Microsoft Windows, and other operating systems use ".." as a path component to indicate a directory level above the current one. On such a system, an HTTP server MUST disallow any such construct in the Request-URI if it would otherwise allow access to a resource outside those intended to be accessible via the HTTP server. Similarly, files intended for reference only internally to the server (such as access control files, configuration files, and script code) MUST be protected from inappropriate retrieval, since they might contain sensitive information. Experience has shown that minor bugs in such HTTP server implementations have turned into security risks.

15.6 Personal Information

HTTP clients are often privy to large amounts of personal information (e.g. the user's name, location, mail address, passwords, encryption keys, etc.), and SHOULD be very careful to prevent unintentional leakage of this information via the HTTP protocol to other sources. We very strongly recommend that a convenient interface be provided for the user to control dissemination of such information, and that designers and implementers be particularly careful in this area. History shows that errors in this area are often both serious security and/or privacy problems, and often generate highly adverse publicity for the implementer's company.

15.7 Privacy Issues Connected to Accept Headers

Accept request-headers can reveal information about the user to all servers which are accessed. The Accept-Language header in particular can reveal information the user would consider to be of a private nature, because the understanding of particular languages is often strongly correlated to the membership of a particular ethnic group. User agents which offer the option to configure the contents of an Accept-Language header to be sent in every request are strongly encouraged to let the configuration process include a message which makes the user aware of the loss of privacy involved.

An approach that limits the loss of privacy would be for a user agent to omit the sending of Accept-Language headers by default, and to ask the user whether it should start sending Accept-Language headers to a server if it detects, by looking for any Vary response-header fields generated by the server, that such sending could improve the quality of service.

Elaborate user-customized accept header fields sent in every request, in particular if these include quality values, can be used by servers as relatively reliable and long-lived user identifiers. Such user identifiers would allow content providers to do click-trail tracking, and would allow collaborating content providers to match cross-server click-trails or form submissions of individual users. Note that for many users not behind a proxy, the network address of the host running the user agent will also serve as a long-lived user identifier. In environments where proxies are used to enhance privacy, user agents should be conservative in offering accept header configuration options to end users. As an extreme privacy measure, proxies could filter the accept headers in relayed requests. General purpose user agents which provide a high degree of header configurability should warn users about the loss of privacy which can be involved.

15.8 DNS Spoofing

Clients using HTTP rely heavily on the Domain Name Service, and are thus generally prone to security attacks based on the deliberate mis-association of IP addresses and DNS names. Clients need to be cautious in assuming the continuing validity of an IP number/DNS name association.

In particular, HTTP clients SHOULD rely on their name resolver for confirmation of an IP number/DNS name association, rather than caching the result of previous host name lookups. Many platforms already can cache host name lookups locally when appropriate, and they SHOULD be configured to do so. These lookups should be cached, however, only when the TTL (Time To Live) information reported by the name server makes it likely that the cached information will remain useful.

If HTTP clients cache the results of host name lookups in order to achieve a performance improvement, they MUST observe the TTL information reported by DNS.

If HTTP clients do not observe this rule, they could be spoofed when a previously-accessed server's IP address changes. As network renumbering is expected to become increasingly common, the possibility of this form of attack will grow. Observing this requirement thus reduces this potential security vulnerability.

This requirement also improves the load-balancing behavior of clients for replicated servers using the same DNS name and reduces the likelihood of a user's experiencing failure in accessing sites which use that strategy.

15.9 Location Headers and Spoofing

If a single server supports multiple organizations that do not trust one another, then it must check the values of Location and Content-Location headers in responses that are generated under control of said organizations to make sure that they do not attempt to invalidate resources over which they have no authority.

16 Acknowledgments

This specification makes heavy use of the augmented BNF and generic constructs defined by David H. Crocker for RFC 822. Similarly, it reuses many of the definitions provided by Nathaniel Borenstein and Ned Freed for MIME. We hope that their inclusion in this specification will help reduce past confusion over the relationship between HTTP and Internet mail message formats.

The HTTP protocol has evolved considerably over the past four years. It has benefited from a large and active developer community--the many people who have participated on the www-talk mailing list--and it is that community which has been most responsible for the success of HTTP and of the World-Wide Web in general. Marc Andreessen, Robert Cailliau, Daniel W. Connolly, Bob Denny, John Franks, Jean-Francois Groff, Phillip M. Hallam-Baker, Hakon W. Lie, Ari Luotonen, Rob McCool, Lou Montulli, Dave Raggett, Tony Sanders, and Marc VanHeyningen deserve special recognition for their efforts in defining early aspects of the protocol.

This document has benefited greatly from the comments of all those participating in the HTTP-WG. In addition to those already mentioned, the following individuals have contributed to this specification:

Gary Adams	Albert Lunde
Harald Tveit Alvestrand	John C. Mallery
Keith Ball	Jean-Philippe Martin-Flatin
Brian Behlendorf	Larry Masinter
Paul Burchard	Mitra
Maurizio Codogno	David Morris
Mike Cowlishaw	Gavin Nicol
Roman Czyborra	Bill Perry
Michael A. Dolan	Jeffrey Perry
David J. Fiander	Scott Powers
Alan Freier	Owen Rees
Marc Hedlund	Luigi Rizzo
Greg Herlihy	David Robinson
Koen Holtman	Marc Salomon
Alex Hopmann	Rich Salz
Bob Jernigan	Allan M. Schiffman
Shel Kaphan	Jim Seidman
Rohit Khare	Chuck Shotton
John Klensin	Eric W. Sink
Martijn Koster	Simon E. Spero
Alexei Kosut	Richard N. Taylor
David M. Kristol	Robert S. Thau
Daniel LaLiberte	Bill (BearHeart) Weinman
Ben Laurie	Francois Yergeau
Paul J. Leach	Mary Ellen Zurko
Daniel DuBois	

Much of the content and presentation of the caching design is due to suggestions and comments from individuals including: Shel Kaphan, Paul Leach, Koen Holtman, David Morris, and Larry Masinter.

Most of the specification of ranges is based on work originally done by Ari Luotonen and John Franks, with additional input from Steve Zilles.

Thanks to the "cave men" of Palo Alto. You know who you are.

Jim Gettys (the current editor of this document) wishes particularly to thank Roy Fielding, the previous editor of this document, along with John Klensin, Jeff Mogul, Paul Leach, Dave Kristol, Koen Holtman, John Franks, Alex Hopmann, and Larry Masinter for their help.

17 References

[1] Alvestrand, H., "Tags for the identification of languages", RFC 1766, UNINETT, March 1995.

[2] Anklesaria, F., McCahill, M., Lindner, P., Johnson, D., Torrey, D., and B. Alberti. "The Internet Gopher Protocol: (a distributed document search and retrieval protocol)", RFC 1436, University of Minnesota, March 1993.

[3] Berners-Lee, T., "Universal Resource Identifiers in WWW", A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web", RFC 1630, CERN, June 1994.

[4] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, CERN, Xerox PARC, University of Minnesota, December 1994.

[5] Berners-Lee, T., and D. Connolly, "HyperText Markup Language Specification - 2.0", RFC 1866, MIT/LCS, November 1995.

[6] Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0.", RFC 1945 MIT/LCS, UC Irvine, May 1996.

[7] Freed, N., and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, Innosoft, First Virtual, November 1996.

[8] Braden, R., "Requirements for Internet hosts - application and support", STD 3, RFC 1123, IETF, October 1989.

[9] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982.

[10] Davis, F., Kahle, B., Morris, H., Salem, J., Shen, T., Wang, R., Sui, J., and M. Grinbaum. "WAIS Interface Protocol Prototype Functional Specification", (v1.5), Thinking Machines Corporation, April 1990.

[11] Fielding, R., "Relative Uniform Resource Locators", RFC 1808, UC Irvine, June 1995.

[12] Horton, M., and R. Adams. "Standard for interchange of USENET messages", RFC 1036, AT&T Bell Laboratories, Center for Seismic Studies, December 1987.

[13] Kantor, B., and P. Lapsley. "Network News Transfer Protocol." A Proposed Standard for the Stream-Based Transmission of News", RFC 977, UC San Diego, UC Berkeley, February 1986.

[14] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, University of Tennessee, November 1996.

[15] Nebel, E., and L. Masinter. "Form-based File Upload in HTML", RFC 1867, Xerox Corporation, November 1995.

[16] Postel, J., "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/ISI, August 1982.

[17] Postel, J., "Media Type Registration Procedure", RFC 2048, USC/ISI, November 1996.

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

[19] Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1700, USC/ISI, October 1994.

[20] Sollins, K., and L. Masinter, "Functional Requirements for Uniform Resource Names", RFC 1737, MIT/LCS, Xerox Corporation, December 1994.

[21] US-ASCII. Coded Character Set - 7-Bit American Standard Code for Information Interchange. Standard ANSI X3.4-1986, ANSI, 1986.

[22] ISO-8859. International Standard -- Information Processing --
8-bit Single-Byte Coded Graphic Character Sets --
Part 1: Latin alphabet No. 1, ISO 8859-1:1987.
Part 2: Latin alphabet No. 2, ISO 8859-2, 1987.
Part 3: Latin alphabet No. 3, ISO 8859-3, 1988.
Part 4: Latin alphabet No. 4, ISO 8859-4, 1988.

Part 5: Latin/Cyrillic alphabet, ISO 8859-5, 1988.
Part 6: Latin/Arabic alphabet, ISO 8859-6, 1987.
Part 7: Latin/Greek alphabet, ISO 8859-7, 1987.
Part 8: Latin/Hebrew alphabet, ISO 8859-8, 1988.
Part 9: Latin alphabet No. 5, ISO 8859-9, 1990.

[23] Meyers, J., and M. Rose "The Content-MD5 Header Field", RFC 1864, Carnegie Mellon, Dover Beach Consulting, October, 1995.

[24] Carpenter, B., and Y. Rekhter, "Renumbering Needs Work", RFC 1900, IAB, February 1996.

[25] Deutsch, P., "GZIP file format specification version 4.3." RFC 1952, Aladdin Enterprises, May 1996.

[26] Venkata N. Padmanabhan and Jeffrey C. Mogul. Improving HTTP Latency. Computer Networks and ISDN Systems, v. 28, pp. 25-35, Dec. 1995. Slightly revised version of paper in Proc. 2nd International WWW Conf. '94: Mosaic and the Web, Oct. 1994, which is available at http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/mogul/HTTPLatency.html.

[27] Joe Touch, John Heidemann, and Katia Obraczka, "Analysis of HTTP Performance", <URL: http://www.isi.edu/lsam/ib/http-perf/>, USC/Information Sciences Institute, June 1996

[28] Mills, D., "Network Time Protocol, Version 3, Specification, Implementation and Analysis", RFC 1305, University of Delaware, March 1992.

[29] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3." RFC 1951, Aladdin Enterprises, May 1996.

[30] Spero, S., "Analysis of HTTP Performance Problems" <URL:http://sunsite.unc.edu/mdma-release/http-prob.html>.

[31] Deutsch, P., and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, Aladdin Enterprises, Info-ZIP, May 1996.

[32] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Sink, E., and L. Stewart, "An Extension to HTTP : Digest Access Authentication", RFC 2069, January 1997.

18 Authors' Addresses

Roy T. Fielding
Department of Information and Computer Science
University of California
Irvine, CA 92717-3425, USA

Fax: +1 (714) 824-4056
EMail: fielding@ics.uci.edu

Jim Gettys
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA

Fax: +1 (617) 258 8682
EMail: jg@w3.org

Jeffrey C. Mogul
Western Research Laboratory
Digital Equipment Corporation
250 University Avenue
Palo Alto, California, 94305, USA

EMail: mogul@wrl.dec.com

Henrik Frystyk Nielsen
W3 Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA

Fax: +1 (617) 258 8682
EMail: frystyk@w3.org

Tim Berners-Lee
Director, W3 Consortium
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139, USA

Fax: +1 (617) 258 8682
EMail: timbl@w3.org

19 Appendices

19.1 Internet Media Type message/http

In addition to defining the HTTP/1.1 protocol, this document serves as the specification for the Internet media type "message/http". The following is to be registered with IANA.

```
    Media Type name:         message
    Media subtype name:      http
    Required parameters:     none
    Optional parameters:     version, msgtype

     version: The HTTP-Version number of the enclosed message
              (e.g., "1.1"). If not present, the version can be
              determined from the first line of the body.

     msgtype: The message type -- "request" or "response". If not
              present, the type can be determined from the first
              line of the body.

    Encoding considerations: only "7bit", "8bit", or "binary" are
                             permitted

    Security considerations: none
```

19.2 Internet Media Type multipart/byteranges

When an HTTP message includes the content of multiple ranges (for example, a response to a request for multiple non-overlapping ranges), these are transmitted as a multipart MIME message. The multipart media type for this purpose is called "multipart/byteranges".

The multipart/byteranges media type includes two or more parts, each with its own Content-Type and Content-Range fields. The parts are separated using a MIME boundary parameter.

```
       Media Type name:         multipart
       Media subtype name:      byteranges
       Required parameters:     boundary
       Optional parameters:     none

       Encoding considerations: only "7bit", "8bit", or "binary" are
                                permitted

       Security considerations: none
```

For example:

```
HTTP/1.1 206 Partial content
Date: Wed, 15 Nov 1995 06:25:24 GMT
Last-modified: Wed, 15 Nov 1995 04:58:08 GMT
Content-type: multipart/byteranges; boundary=THIS_STRING_SEPARATES

--THIS_STRING_SEPARATES
Content-type: application/pdf
Content-range: bytes 500-999/8000

...the first range...
--THIS_STRING_SEPARATES
Content-type: application/pdf
Content-range: bytes 7000-7999/8000

...the second range
--THIS_STRING_SEPARATES--
```

19.3 Tolerant Applications

Although this document specifies the requirements for the generation of HTTP/1.1 messages, not all applications will be correct in their implementation. We therefore recommend that operational applications be tolerant of deviations whenever those deviations can be interpreted unambiguously.

Clients SHOULD be tolerant in parsing the Status-Line and servers tolerant when parsing the Request-Line. In particular, they SHOULD accept any amount of SP or HT characters between fields, even though only a single SP is required.

The line terminator for message-header fields is the sequence CRLF. However, we recommend that applications, when parsing such headers, recognize a single LF as a line terminator and ignore the leading CR.

The character set of an entity-body should be labeled as the lowest common denominator of the character codes used within that body, with the exception that no label is preferred over the labels US-ASCII or ISO-8859-1.

Additional rules for requirements on parsing and encoding of dates and other potential problems with date encodings include:

- HTTP/1.1 clients and caches should assume that an RFC-850 date which appears to be more than 50 years in the future is in fact in the past (this helps solve the "year 2000" problem).

- An HTTP/1.1 implementation may internally represent a parsed Expires date as earlier than the proper value, but MUST NOT internally represent a parsed Expires date as later than the proper value.

- All expiration-related calculations must be done in GMT. The local time zone MUST NOT influence the calculation or comparison of an age or expiration time.

- If an HTTP header incorrectly carries a date value with a time zone other than GMT, it must be converted into GMT using the most conservative possible conversion.

19.4 Differences Between HTTP Entities and MIME Entities

HTTP/1.1 uses many of the constructs defined for Internet Mail (RFC 822) and the Multipurpose Internet Mail Extensions (MIME) to allow entities to be transmitted in an open variety of representations and with extensible mechanisms. However, MIME [7] discusses mail, and HTTP has a few features that are different from those described in MIME. These differences were carefully chosen to optimize performance over binary connections, to allow greater freedom in the use of new media types, to make date comparisons easier, and to acknowledge the practice of some early HTTP servers and clients.

This appendix describes specific areas where HTTP differs from MIME. Proxies and gateways to strict MIME environments SHOULD be aware of these differences and provide the appropriate conversions where necessary. Proxies and gateways from MIME environments to HTTP also need to be aware of the differences because some conversions may be required.

19.4.1 Conversion to Canonical Form

MIME requires that an Internet mail entity be converted to canonical form prior to being transferred. Section 3.7.1 of this document describes the forms allowed for subtypes of the "text" media type when transmitted over HTTP. MIME requires that content with a type of "text" represent line breaks as CRLF and forbids the use of CR or LF outside of line break sequences. HTTP allows CRLF, bare CR, and bare LF to indicate a line break within text content when a message is transmitted over HTTP.

Where it is possible, a proxy or gateway from HTTP to a strict MIME environment SHOULD translate all line breaks within the text media types described in section 3.7.1 of this document to the MIME canonical form of CRLF. Note, however, that this may be complicated by the presence of a Content-Encoding and by the fact that HTTP

allows the use of some character sets which do not use octets 13 and 10 to represent CR and LF, as is the case for some multi-byte character sets.

19.4.2 Conversion of Date Formats

HTTP/1.1 uses a restricted set of date formats (section 3.3.1) to simplify the process of date comparison. Proxies and gateways from other protocols SHOULD ensure that any Date header field present in a message conforms to one of the HTTP/1.1 formats and rewrite the date if necessary.

19.4.3 Introduction of Content-Encoding

MIME does not include any concept equivalent to HTTP/1.1's Content-Encoding header field. Since this acts as a modifier on the media type, proxies and gateways from HTTP to MIME-compliant protocols MUST either change the value of the Content-Type header field or decode the entity-body before forwarding the message. (Some experimental applications of Content-Type for Internet mail have used a media-type parameter of ";conversions=<content-coding>" to perform an equivalent function as Content-Encoding. However, this parameter is not part of MIME.)

19.4.4 No Content-Transfer-Encoding

HTTP does not use the Content-Transfer-Encoding (CTE) field of MIME. Proxies and gateways from MIME-compliant protocols to HTTP MUST remove any non-identity CTE ("quoted-printable" or "base64") encoding prior to delivering the response message to an HTTP client.

Proxies and gateways from HTTP to MIME-compliant protocols are responsible for ensuring that the message is in the correct format and encoding for safe transport on that protocol, where "safe transport" is defined by the limitations of the protocol being used. Such a proxy or gateway SHOULD label the data with an appropriate Content-Transfer-Encoding if doing so will improve the likelihood of safe transport over the destination protocol.

19.4.5 HTTP Header Fields in Multipart Body-Parts

In MIME, most header fields in multipart body-parts are generally ignored unless the field name begins with "Content-". In HTTP/1.1, multipart body-parts may contain any HTTP header fields which are significant to the meaning of that part.

19.4.6 Introduction of Transfer-Encoding

HTTP/1.1 introduces the Transfer-Encoding header field (section 14.40). Proxies/gateways MUST remove any transfer coding prior to forwarding a message via a MIME-compliant protocol.

A process for decoding the "chunked" transfer coding (section 3.6) can be represented in pseudo-code as:

```
length := 0
read chunk-size, chunk-ext (if any) and CRLF
while (chunk-size > 0) {
   read chunk-data and CRLF
   append chunk-data to entity-body
   length := length + chunk-size
   read chunk-size and CRLF
}
read entity-header
while (entity-header not empty) {
   append entity-header to existing header fields
   read entity-header
}
Content-Length := length
Remove "chunked" from Transfer-Encoding
```

19.4.7 MIME-Version

HTTP is not a MIME-compliant protocol (see appendix 19.4). However, HTTP/1.1 messages may include a single MIME-Version general-header field to indicate what version of the MIME protocol was used to construct the message. Use of the MIME-Version header field indicates that the message is in full compliance with the MIME protocol. Proxies/gateways are responsible for ensuring full compliance (where possible) when exporting HTTP messages to strict MIME environments.

```
MIME-Version   = "MIME-Version" ":" 1*DIGIT "." 1*DIGIT
```

MIME version "1.0" is the default for use in HTTP/1.1. However, HTTP/1.1 message parsing and semantics are defined by this document and not the MIME specification.

19.5 Changes from HTTP/1.0

This section summarizes major differences between versions HTTP/1.0 and HTTP/1.1.

19.5.1 Changes to Simplify Multi-homed Web Servers and Conserve IP Addresses

The requirements that clients and servers support the Host request-header, report an error if the Host request-header (section 14.23) is missing from an HTTP/1.1 request, and accept absolute URIs (section 5.1.2) are among the most important changes defined by this specification.

Older HTTP/1.0 clients assumed a one-to-one relationship of IP addresses and servers; there was no other established mechanism for distinguishing the intended server of a request than the IP address to which that request was directed. The changes outlined above will allow the Internet, once older HTTP clients are no longer common, to support multiple Web sites from a single IP address, greatly simplifying large operational Web servers, where allocation of many IP addresses to a single host has created serious problems. The Internet will also be able to recover the IP addresses that have been allocated for the sole purpose of allowing special-purpose domain names to be used in root-level HTTP URLs. Given the rate of growth of the Web, and the number of servers already deployed, it is extremely important that all implementations of HTTP (including updates to existing HTTP/1.0 applications) correctly implement these requirements:

- Both clients and servers MUST support the Host request-header.
- Host request-headers are required in HTTP/1.1 requests.
- Servers MUST report a 400 (Bad Request) error if an HTTP/1.1 request does not include a Host request-header.
- Servers MUST accept absolute URIs.

19.6 Additional Features

This appendix documents protocol elements used by some existing HTTP implementations, but not consistently and correctly across most HTTP/1.1 applications. Implementers should be aware of these features, but cannot rely upon their presence in, or interoperability with, other HTTP/1.1 applications. Some of these describe proposed experimental features, and some describe features that experimental deployment found lacking that are now addressed in the base HTTP/1.1 specification.

19.6.1 Additional Request Methods

19.6.1.1 PATCH

The PATCH method is similar to PUT except that the entity contains a list of differences between the original version of the resource identified by the Request-URI and the desired content of the resource after the PATCH action has been applied. The list of differences is in a format defined by the media type of the entity (e.g., "application/diff") and MUST include sufficient information to allow the server to recreate the changes necessary to convert the original version of the resource to the desired version.

If the request passes through a cache and the Request-URI identifies a currently cached entity, that entity MUST be removed from the cache. Responses to this method are not cachable.

The actual method for determining how the patched resource is placed, and what happens to its predecessor, is defined entirely by the origin server. If the original version of the resource being patched included a Content-Version header field, the request entity MUST include a Derived-From header field corresponding to the value of the original Content-Version header field. Applications are encouraged to use these fields for constructing versioning relationships and resolving version conflicts.

PATCH requests must obey the message transmission requirements set out in section 8.2.

Caches that implement PATCH should invalidate cached responses as defined in section 13.10 for PUT.

19.6.1.2 LINK

The LINK method establishes one or more Link relationships between the existing resource identified by the Request-URI and other existing resources. The difference between LINK and other methods

allowing links to be established between resources is that the LINK method does not allow any message-body to be sent in the request and does not directly result in the creation of new resources.

If the request passes through a cache and the Request-URI identifies a currently cached entity, that entity MUST be removed from the cache. Responses to this method are not cachable.

Caches that implement LINK should invalidate cached responses as defined in section 13.10 for PUT.

19.6.1.3 UNLINK

The UNLINK method removes one or more Link relationships from the existing resource identified by the Request-URI. These relationships may have been established using the LINK method or by any other method supporting the Link header. The removal of a link to a resource does not imply that the resource ceases to exist or becomes inaccessible for future references.

If the request passes through a cache and the Request-URI identifies a currently cached entity, that entity MUST be removed from the cache. Responses to this method are not cachable.

Caches that implement UNLINK should invalidate cached responses as defined in section 13.10 for PUT.

19.6.2 Additional Header Field Definitions

19.6.2.1 Alternates

The Alternates response-header field has been proposed as a means for the origin server to inform the client about other available representations of the requested resource, along with their distinguishing attributes, and thus providing a more reliable means for a user agent to perform subsequent selection of another representation which better fits the desires of its user (described as agent-driven negotiation in section 12).

The Alternates header field is orthogonal to the Vary header field in that both may coexist in a message without affecting the interpretation of the response or the available representations. It is expected that Alternates will provide a significant improvement over the server-driven negotiation provided by the Vary field for those resources that vary over common dimensions like type and language.

The Alternates header field will be defined in a future specification.

19.6.2.2 Content-Version

The Content-Version entity-header field defines the version tag associated with a rendition of an evolving entity. Together with the Derived-From field described in section 19.6.2.3, it allows a group of people to work simultaneously on the creation of a work as an iterative process. The field should be used to allow evolution of a particular work along a single path rather than derived works or renditions in different representations.

```
Content-Version = "Content-Version" ":" quoted-string
```

Examples of the Content-Version field include:

```
Content-Version: "2.1.2"
Content-Version: "Fred 19950116-12:26:48"
Content-Version: "2.5a4-omega7"
```

19.6.2.3 Derived-From

The Derived-From entity-header field can be used to indicate the version tag of the resource from which the enclosed entity was derived before modifications were made by the sender. This field is used to help manage the process of merging successive changes to a resource, particularly when such changes are being made in parallel and from multiple sources.

```
Derived-From   = "Derived-From" ":" quoted-string
```

An example use of the field is:

```
Derived-From: "2.1.1"
```

The Derived-From field is required for PUT and PATCH requests if the entity being sent was previously retrieved from the same URI and a Content-Version header was included with the entity when it was last retrieved.

19.6.2.4 Link

The Link entity-header field provides a means for describing a relationship between two resources, generally between the requested resource and some other resource. An entity MAY include multiple Link values. Links at the metainformation level typically indicate relationships like hierarchical structure and navigation paths. The Link field is semantically equivalent to the <LINK> element in HTML.[5]

```
       Link           = "Link" ":" #("<" URI ">" *( ";" link-param )

       link-param     = ( ( "rel" "=" relationship )
                        | ( "rev" "=" relationship )
                        | ( "title" "=" quoted-string )
                        | ( "anchor" "=" <"> URI <"> )
                        | ( link-extension ) )

       link-extension = token [ "=" ( token | quoted-string ) ]

       relationship   = sgml-name
                      | ( <"> sgml-name *( SP sgml-name) <"> )

       sgml-name      = ALPHA *( ALPHA | DIGIT | "." | "-" )
```

Relationship values are case-insensitive and MAY be extended within the constraints of the sgml-name syntax. The title parameter MAY be used to label the destination of a link such that it can be used as identification within a human-readable menu. The anchor parameter MAY be used to indicate a source anchor other than the entire current resource, such as a fragment of this resource or a third resource.

Examples of usage include:

```
    Link: <http://www.cern.ch/TheBook/chapter2>; rel="Previous"

    Link: <mailto:timbl@w3.org>; rev="Made"; title="Tim Berners-Lee"
```

The first example indicates that chapter2 is previous to this resource in a logical navigation path. The second indicates that the person responsible for making the resource available is identified by the given e-mail address.

19.6.2.5 URI

The URI header field has, in past versions of this specification, been used as a combination of the existing Location, Content-Location, and Vary header fields as well as the future Alternates field (above). Its primary purpose has been to include a list of additional URIs for the resource, including names and mirror locations. However, it has become clear that the combination of many different functions within this single field has been a barrier to consistently and correctly implementing any of those functions. Furthermore, we believe that the identification of names and mirror locations would be better performed via the Link header field. The URI header field is therefore deprecated in favor of those other fields.

```
       URI-header    = "URI" ":" 1#( "<" URI ">" )
```

19.7 Compatibility with Previous Versions

It is beyond the scope of a protocol specification to mandate compliance with previous versions. HTTP/1.1 was deliberately designed, however, to make supporting previous versions easy. It is worth noting that at the time of composing this specification, we would expect commercial HTTP/1.1 servers to:

- recognize the format of the Request-Line for HTTP/0.9, 1.0, and 1.1 requests;
- understand any valid request in the format of HTTP/0.9, 1.0, or 1.1;
- respond appropriately with a message in the same major version used by the client.

And we would expect HTTP/1.1 clients to:

- recognize the format of the Status-Line for HTTP/1.0 and 1.1 responses;
- understand any valid response in the format of HTTP/0.9, 1.0, or 1.1.

For most implementations of HTTP/1.0, each connection is established by the client prior to the request and closed by the server after sending the response. A few implementations implement the Keep-Alive version of persistent connections described in section 19.7.1.1.

19.7.1 Compatibility with HTTP/1.0 Persistent Connections

Some clients and servers may wish to be compatible with some previous implementations of persistent connections in HTTP/1.0 clients and servers. Persistent connections in HTTP/1.0 must be explicitly negotiated as they are not the default behavior. HTTP/1.0 experimental implementations of persistent connections are faulty, and the new facilities in HTTP/1.1 are designed to rectify these problems. The problem was that some existing 1.0 clients may be sending Keep-Alive to a proxy server that doesn't understand Connection, which would then erroneously forward it to the next inbound server, which would establish the Keep-Alive connection and result in a hung HTTP/1.0 proxy waiting for the close on the response. The result is that HTTP/1.0 clients must be prevented from using Keep-Alive when talking to proxies.

However, talking to proxies is the most important use of persistent connections, so that prohibition is clearly unacceptable. Therefore, we need some other mechanism for indicating a persistent connection is desired, which is safe to use even when talking to an old proxy that ignores Connection. Persistent connections are the default for HTTP/1.1 messages; we introduce a new keyword (Connection: close) for declaring non-persistence.

The following describes the original HTTP/1.0 form of persistent connections.

When it connects to an origin server, an HTTP client MAY send the Keep-Alive connection-token in addition to the Persist connection-token:

```
Connection: Keep-Alive
```

An HTTP/1.0 server would then respond with the Keep-Alive connection token and the client may proceed with an HTTP/1.0 (or Keep-Alive) persistent connection.

An HTTP/1.1 server may also establish persistent connections with HTTP/1.0 clients upon receipt of a Keep-Alive connection token. However, a persistent connection with an HTTP/1.0 client cannot make use of the chunked transfer-coding, and therefore MUST use a Content-Length for marking the ending boundary of each message.

A client MUST NOT send the Keep-Alive connection token to a proxy server as HTTP/1.0 proxy servers do not obey the rules of HTTP/1.1 for parsing the Connection header field.

19.7.1.1 The Keep-Alive Header

When the Keep-Alive connection-token has been transmitted with a request or a response, a Keep-Alive header field MAY also be included. The Keep-Alive header field takes the following form:

```
Keep-Alive-header = "Keep-Alive" ":" 0# keepalive-param

keepalive-param = param-name "=" value
```

The Keep-Alive header itself is optional, and is used only if a parameter is being sent. HTTP/1.1 does not define any parameters.

If the Keep-Alive header is sent, the corresponding connection token MUST be transmitted. The Keep-Alive header MUST be ignored if received without the connection token.

Network Working Group
Request for Comments: 977

Brian Kantor (U.C. San Diego)
Phil Lapsley (U.C. Berkeley)

February 1986

Network News Transfer Protocol

A Proposed Standard for the Stream-Based Transmission of News

Status of This Memo

NNTP specifies a protocol for the distribution, inquiry, retrieval, and posting of news articles using a reliable stream-based transmission of news among the ARPA-Internet community. NNTP is designed so that news articles are stored in a central database allowing a subscriber to select only those items he wishes to read. Indexing, cross-referencing, and expiration of aged messages are also provided. This RFC suggests a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

1. Introduction

For many years, the ARPA-Internet community has supported the distribution of bulletins, information, and data in a timely fashion to thousands of participants. We collectively refer to such items of information as "news". Such news provides for the rapid dissemination of items of interest such as software bug fixes, new product reviews, technical tips, and programming pointers, as well as rapid-fire discussions of matters of concern to the working computer professional. News is very popular among its readers.

There are popularly two methods of distributing such news: the Internet method of direct mailing, and the USENET news system.

1.1. Internet Mailing Lists

The Internet community distributes news by the use of mailing lists. These are lists of subscriber's mailbox addresses and remailing sublists of all intended recipients. These mailing lists operate by remailing a copy of the information to be distributed to each subscriber on the mailing list. Such remailing is inefficient when a mailing list grows beyond a dozen or so people, since sending a separate copy to each of the subscribers occupies large quantities of network bandwidth, CPU resources, and significant amounts of disk storage at the destination host. There is also a significant problem in maintenance of the list itself: as subscribers move from one job to another; as new subscribers join and old ones leave; and as hosts come in and out of service.

1.2. The USENET News System

Clearly, a worthwhile reduction of the amount of these resources used can be achieved if articles are stored in a central database on the receiving host instead of in each subscriber's mailbox. The USENET news system provides a method of doing just this. There is a central repository of the news articles in one place (customarily a spool directory of some sort), and a set of programs that allow a subscriber to select those items he wishes to read. Indexing, cross-referencing, and expiration of aged messages are also provided.

1.3. Central Storage of News

For clusters of hosts connected together by fast local area networks (such as Ethernet), it makes even more sense to consolidate news distribution onto one (or a very few) hosts, and to allow access to these news articles using a server and client model. Subscribers may then request only the articles they wish to see, without having to wastefully duplicate the storage of a copy of each item on each host.

1.4. A Central News Server

A way to achieve these economies is to have a central computer system that can provide news service to the other systems on the local area network. Such a server would manage the collection of news articles and index files, with each person who desires to read news bulletins doing so over the LAN. For a large cluster of computer systems, the savings in total disk space is clearly worthwhile. Also, this allows workstations with limited disk storage space to participate in the news without incoming items consuming oppressive amounts of the workstation's disk storage.

We have heard rumors of somewhat successful attempts to provide centralized news service using IBIS and other shared or distributed file systems. While it is possible that such a distributed file system implementation might work well with a group of similar computers running nearly identical operating systems, such a scheme is not general enough to offer service to a wide range of client systems, especially when many diverse operating systems may be in use among a group of clients. There are few (if any) shared or networked file systems that can offer the generality of service that stream connections using Internet TCP provide, particularly when a wide range of host hardware and operating systems are considered.

NNTP specifies a protocol for the distribution, inquiry, retrieval, and posting of news articles using a reliable stream (such as TCP) server-client model. NNTP is designed so that news articles need only

be stored on one (presumably central) host, and subscribers on other hosts attached to the LAN may read news articles using stream connections to the news host.

NNTP is modelled upon the news article specifications in RFC 850, which describes the USENET news system. However, NNTP makes few demands upon the structure, content, or storage of news articles, and thus we believe it easily can be adapted to other non-USENET news systems.

Typically, the NNTP server runs as a background process on one host, and would accept connections from other hosts on the LAN. This works well when there are a number of small computer systems (such as workstations, with only one or at most a few users each), and a large central server.

1.5. Intermediate News Servers

For clusters of machines with many users (as might be the case in a university or large industrial environment), an intermediate server might be used. This intermediate or "slave" server runs on each computer system, and is responsible for mediating news reading requests and performing local caching of recently-retrieved news articles.

Typically, a client attempting to obtain news service would first attempt to connect to the news service port on the local machine. If this attempt were unsuccessful, indicating a failed server, an installation might choose to either deny news access, or to permit connection to the central "master" news server.

For workstations or other small systems, direct connection to the master server would probably be the normal manner of operation.

This specification does not cover the operation of slave NNTP servers. We merely suggest that slave servers are a logical addition to NNTP server usage which would enhance operation on large local area networks.

1.6. News Distribution

NNTP has commands which provide a straightforward method of exchanging articles between cooperating hosts. Hosts which are well connected on a local area or other fast network and who wish to actually obtain copies of news articles for local storage might well find NNTP to be a more efficient way to distribute news than more traditional transfer methods (such as UUCP).

In the traditional method of distributing news articles, news is propagated from host to host by flooding - that is, each host will send all its new news articles on to each host that it feeds. These hosts will then in turn send these new articles on to other hosts that they feed. Clearly, sending articles that a host already has obtained a copy of from another feed (many hosts that receive news are redundantly fed) again is a waste of time and communications resources, but for transport mechanisms that are single-transaction based rather than interactive (such as UUCP in the UNIX-world <1>), distribution time is diminished by sending all articles and having the receiving host simply discard the duplicates. This is an especially true when communications sessions are limited to once a day.

Using NNTP, hosts exchanging news articles have an interactive mechanism for deciding which articles are to be transmitted. A host desiring new news, or which has new news to send, will typically contact one or more of its neighbors using NNTP. First it will inquire if any new news groups have been created on the serving host by means of the NEWGROUPS command. If so, and those are appropriate or desired (as established by local site-dependent rules), those new newsgroups can be created.

The client host will then inquire as to which new articles have arrived in all or some of the newsgroups that it desires to receive, using the NEWNEWS command. It will receive a list of new articles from the server, and can request transmission of those articles that it desires and does not already have.

Finally, the client can advise the server of those new articles which the client has recently received. The server will indicate those articles that it has already obtained copies of, and which articles should be sent to add to its collection.

In this manner, only those articles which are not duplicates and which are desired are transferred.

2. The NNTP Specification

2.1. Overview

The news server specified by this document uses a stream connection (such as TCP) and SMTP-like commands and responses. It is designed to accept connections from hosts, and to provide a simple interface to the news database.

This server is only an interface between programs and the news databases. It does not perform any user interaction or presentation-level functions. These "user-friendly" functions are better left to the client programs, which have a better understanding of the environment in which they are operating.

When used via Internet TCP, the contact port assigned for this service is 119.

2.2. Character Codes

Commands and replies are composed of characters from the ASCII character set. When the transport service provides an 8-bit byte (octet) transmission channel, each 7-bit character is transmitted right justified in an octet with the high order bit cleared to zero.

2.3. Commands

Commands consist of a command word, which in some cases may be followed by a parameter. Commands with parameters must separate the parameters from each other and from the command by one or more space or tab characters. Command lines must be complete with all required parameters, and may not contain more than one command.

Commands and command parameters are not case sensitive. That is, a command or parameter word may be upper case, lower case, or any mixture of upper and lower case.

Each command line must be terminated by a CR-LF (Carriage Return - Line Feed) pair.

Command lines shall not exceed 512 characters in length, counting all characters including spaces, separators, punctuation, and the trailing CR-LF (thus there are 510 characters maximum allowed for the command and its parameters). There is no provision for continuation command lines.

2.4. Responses

Responses are of two kinds, textual and status.

2.4.1. Text Responses

Text is sent only after a numeric status response line has been sent that indicates that text will follow. Text is sent as a series of successive lines of textual matter, each terminated with CR-LF pair. A single line containing only a period (.) is sent to indicate the end of the text (i.e., the server will send a CR-LF pair at the end of the last line of text, a period, and another CR-LF pair).

If the text contained a period as the first character of the text line in the original, that first period is doubled. Therefore, the client must examine the first character of each line received, and for those beginning with a period, determine either that this is the end of the text or whether to collapse the doubled period to a single one.

The intention is that text messages will usually be displayed on the user's terminal whereas command/status responses will be interpreted by the client program before any possible display is done.

2.4.2. Status Responses

These are status reports from the server and indicate the response to the last command received from the client.

Status response lines begin with a 3 digit numeric code which is sufficient to distinguish all responses. Some of these may herald the subsequent transmission of text.

The first digit of the response broadly indicates the success, failure, or progress of the previous command.

```
   1xx - Informative message
   2xx - Command ok
   3xx - Command ok so far, send the rest of it.
   4xx - Command was correct, but couldn't be performed for
         some reason.
   5xx - Command unimplemented, or incorrect, or a serious
         program error occurred.
```

The next digit in the code indicates the function response category.

```
x0x - Connection, setup, and miscellaneous messages
x1x - Newsgroup selection
x2x - Article selection
x3x - Distribution functions
x4x - Posting
x8x - Nonstandard (private implementation) extensions
x9x - Debugging output
```

The exact response codes that should be expected from each command are detailed in the description of that command. In addition, below is listed a general set of response codes that may be received at any time.

Certain status responses contain parameters such as numbers and names. The number and type of such parameters is fixed for each response code to simplify interpretation of the response.

Parameters are separated from the numeric response code and from each other by a single space. All numeric parameters are decimal, and may have leading zeros. All string parameters begin after the separating space, and end before the following separating space or the CR-LF pair at the end of the line. (String parameters may not, therefore, contain spaces.) All text, if any, in the response which is not a parameter of the response must follow and be separated from the last parameter by a space. Also, note that the text following a response number may vary in different implementations of the server. The 3-digit numeric code should be used to determine what response was sent.

Response codes not specified in this standard may be used for any installation-specific additional commands also not specified. These should be chosen to fit the pattern of x8x specified above. (Note that debugging is provided for explicitly in the x9x response codes.) The use of unspecified response codes for standard commands is prohibited.

We have provided a response pattern x9x for debugging. Since much debugging output may be classed as "informative messages", we would expect, therefore, that responses 190 through 199 would be used for various debugging outputs. There is no requirement in this specification for debugging output, but if such is provided over the connected stream, it must use these response codes. If appropriate to a specific implementation, other x9x codes may be used for debugging. (An example might be to use e.g., 290 to acknowledge a remote debugging request.)

2.4.3. General Responses

The following is a list of general response codes that may be sent by the NNTP server. These are not specific to any one command, but may be returned as the result of a connection, a failure, or some unusual condition.

In general, 1xx codes may be ignored or displayed as desired; code 200 or 201 is sent upon initial connection to the NNTP server depending upon posting permission; code 400 will be sent when the NNTP server discontinues service (by operator request, for example); and 5xx codes indicate that the command could not be performed for some unusual reason.

```
100 help text
190
  through
199 debug output

200 server ready - posting allowed
201 server ready - no posting allowed

400 service discontinued

500 command not recognized
501 command syntax error
502 access restriction or permission denied
503 program fault - command not performed
```

3. Command and Response Details

On the following pages are descriptions of each command recognized by the NNTP server and the responses which will be returned by those commands.

Each command is shown in upper case for clarity, although case is ignored in the interpretation of commands by the NNTP server. Any parameters are shown in lower case. A parameter shown in [square brackets] is optional. For example, [GMT] indicates that the triglyph GMT may present or omitted.

Every command described in this section must be implemented by all NNTP servers.

There is no prohibition against additional commands being added; however, it is recommended that any such unspecified command begin with the letter "X" to avoid conflict with later revisions of this specification.

Implementors are reminded that such additional commands may not redefine specified status response codes. Using additional unspecified responses for standard commands is also prohibited.

3.1. The ARTICLE, BODY, HEAD, and STAT commands

There are two forms to the ARTICLE command (and the related BODY, HEAD, and STAT commands), each using a different method of specifying which article is to be retrieved. When the ARTICLE command is followed by a message-id in angle brackets ("<" and ">"), the first form of the command is used; when a numeric parameter or no parameter is supplied, the second form is invoked.

The text of the article is returned as a textual response, as described earlier in this document.

The HEAD and BODY commands are identical to the ARTICLE command except that they respectively return only the header lines or text body of the article.

The STAT command is similar to the ARTICLE command except that no text is returned. When selecting by message number within a group, the STAT command serves to set the current article pointer without sending text. The returned acknowledgement response will contain the message-id, which may be of some value. Using the STAT command to select by message-id is valid but of questionable value, since a selection by message-id does NOT alter the "current article pointer".

3.1.1. ARTICLE (selection by message-id)

ARTICLE <message-id>

Display the header, a blank line, then the body (text) of the specified article. Message-id is the message id of an article as shown in that article's header. It is anticipated that the client will obtain the message-id from a list provided by the NEWNEWS command, from references contained within another article, or from the message-id provided in the response to some other commands.

Please note that the internally-maintained "current article pointer" is NOT ALTERED by this command. This is both to facilitate the presentation of articles that may be referenced within an article

being read, and because of the semantic difficulties of determining the proper sequence and membership of an article which may have been posted to more than one newsgroup.

3.1.2. ARTICLE (selection by number)

ARTICLE [nnn]

Displays the header, a blank line, then the body (text) of the current or specified article. The optional parameter nnn is the

numeric id of an article in the current newsgroup and must be chosen from the range of articles provided when the newsgroup was selected. If it is omitted, the current article is assumed.

The internally-maintained "current article pointer" is set by this command if a valid article number is specified.

[the following applies to both forms of the article command.] A response indicating the current article number, a message-id string, and that text is to follow will be returned.

The message-id string returned is an identification string contained within angle brackets ("<" and ">"), which is derived from the header of the article itself. The Message-ID header line (required by RFC850) from the article must be used to supply this information. If the message-id header line is missing from the article, a single digit "0" (zero) should be supplied within the angle brackets.

Since the message-id field is unique with each article, it may be used by a news reading program to skip duplicate displays of articles that have been posted more than once, or to more than one newsgroup.

3.1.3. Responses

220 n <a> article retrieved - head and body follow
(n = article number, <a> = message-id)
221 n <a> article retrieved - head follows
222 n <a> article retrieved - body follows
223 n <a> article retrieved - request text separately
412 no newsgroup has been selected
420 no current article has been selected
423 no such article number in this group
430 no such article found

3.2. The GROUP command

3.2.1. GROUP

GROUP ggg

The required parameter ggg is the name of the newsgroup to be selected (e.g. "net.news"). A list of valid newsgroups may be obtained from the LIST command.

The successful selection response will return the article numbers of the first and last articles in the group, and an estimate of the number of articles on file in the group. It is not necessary that the estimate be correct, although that is helpful; it must only be equal to or larger than the actual number of articles on file. (Some implementations will actually count the number of articles on file. Others will just subtract first article number from last to get an estimate.)

When a valid group is selected by means of this command, the internally maintained "current article pointer" is set to the first article in the group. If an invalid group is specified, the previously selected group and article remain selected. If an empty newsgroup is selected, the "current article pointer" is in an indeterminate state and should not be used.

Note that the name of the newsgroup is not case-dependent. It must otherwise match a newsgroup obtained from the LIST command or an error will result.

3.2.2. Responses

```
211 n f l s group selected
        (n = estimated number of articles in group,
        f = first article number in the group,
        l = last article number in the group,
        s = name of the group.)
411 no such news group
```


3.3. The HELP command

3.3.1. HELP

HELP

Provides a short summary of commands that are understood by this implementation of the server. The help text will be presented as a textual response, terminated by a single period on a line by itself.

3.3.2. Responses

100 help text follows

3.4. The IHAVE command

3.4.1. IHAVE

IHAVE <messageid>

The IHAVE command informs the server that the client has an article whose id is <messageid>. If the server desires a copy of that article, it will return a response instructing the client to send the entire article. If the server does not want the article (if, for example, the server already has a copy of it), a response indicating that the article is not wanted will be returned.

If transmission of the article is requested, the client should send the entire article, including header and body, in the manner specified for text transmission from the server. A response code indicating success or failure of the transferral of the article will be returned.

This function differs from the POST command in that it is intended for use in transferring already-posted articles between hosts. Normally it will not be used when the client is a personal newsreading program. In particular, this function will invoke the server's news posting program with the appropriate settings (flags, options, etc) to indicate that the forthcoming article is being forwarded from another host.

The server may, however, elect not to post or forward the article if after further examination of the article it deems it inappropriate to do so. The 436 or 437 error codes may be returned as appropriate to the situation.

Reasons for such subsequent rejection of an article may include such

problems as inappropriate newsgroups or distributions, disk space limitations, article lengths, garbled headers, and the like. These are typically restrictions enforced by the server host's news software and not necessarily the NNTP server itself.

3.4.2. Responses

235 article transferred ok
335 send article to be transferred. End with <CR-LF>.<CR-LF>
435 article not wanted - do not send it
436 transfer failed - try again later
437 article rejected - do not try again

An implementation note:

Because some host news posting software may not be able to decide immediately that an article is inappropriate for posting or forwarding, it is acceptable to acknowledge the successful transfer of the article and to later silently discard it. Thus it is permitted to return the 235 acknowledgement code and later discard the received article. This is not a fully satisfactory solution to the problem. Perhaps some implementations will wish to send mail to the author of the article in certain of these cases.

3.5. The LAST command

3.5.1. LAST

LAST

The internally maintained "current article pointer" is set to the previous article in the current newsgroup. If already positioned at the first article of the newsgroup, an error message is returned and the current article remains selected.

The internally-maintained "current article pointer" is set by this command.

A response indicating the current article number, and a message-id string will be returned. No text is sent in response to this command.

3.5.2. Responses

223 n a article retrieved - request text separately
(n = article number, a = unique article id)

412 no newsgroup selected
420 no current article has been selected
422 no previous article in this group

3.6. The LIST command

3.6.1. LIST

LIST

Returns a list of valid newsgroups and associated information. Each newsgroup is sent as a line of text in the following format:

group last first p

where <group> is the name of the newsgroup, <last> is the number of the last known article currently in that newsgroup, <first> is the number of the first article currently in the newsgroup, and <p> is either 'y' or 'n' indicating whether posting to this newsgroup is allowed ('y') or prohibited ('n').

The <first> and <last> fields will always be numeric. They may have leading zeros. If the <last> field evaluates to less than the <first> field, there are no articles currently on file in the newsgroup.

Note that posting may still be prohibited to a client even though the LIST command indicates that posting is permitted to a particular newsgroup. See the POST command for an explanation of client prohibitions. The posting flag exists for each newsgroup because some newsgroups are moderated or are digests, and therefore cannot be posted to; that is, articles posted to them must be mailed to a moderator who will post them for the submitter. This is independent of the posting permission granted to a client by the NNTP server.

Please note that an empty list (i.e., the text body returned by this command consists only of the terminating period) is a possible valid response, and indicates that there are currently no valid newsgroups.

3.6.2. Responses

215 list of newsgroups follows

3.7. The NEWGROUPS command

3.7.1. NEWGROUPS

NEWGROUPS date time [GMT] [<distributions>]

A list of newsgroups created since <date and time> will be listed in the same format as the LIST command.

The date is sent as 6 digits in the format YYMMDD, where YY is the last two digits of the year, MM is the two digits of the month (with leading zero, if appropriate), and DD is the day of the month (with leading zero, if appropriate). The closest century is assumed as part of the year (i.e., 86 specifies 1986, 30 specifies 2030, 99 is 1999, 00 is 2000).

Time must also be specified. It must be as 6 digits HHMMSS with HH being hours on the 24-hour clock, MM minutes 00-59, and SS seconds 00-59. The time is assumed to be in the server's timezone unless the token "GMT" appears, in which case both time and date are evaluated at the 0 meridian.

The optional parameter "distributions" is a list of distribution groups, enclosed in angle brackets. If specified, the distribution portion of a new newsgroup (e.g, 'net' in 'net.wombat') will be examined for a match with the distribution categories listed, and only those new newsgroups which match will be listed. If more than one distribution group is to be listed, they must be separated by commas within the angle brackets.

Please note that an empty list (i.e., the text body returned by this command consists only of the terminating period) is a possible valid response, and indicates that there are currently no new newsgroups.

3.7.2. Responses

231 list of new newsgroups follows

3.8. The NEWNEWS command

3.8.1. NEWNEWS

NEWNEWS newsgroups date time [GMT] [<distribution>]

A list of message-ids of articles posted or received to the specified newsgroup since "date" will be listed. The format of the listing will be one message-id per line, as though text were being sent. A single line consisting solely of one period followed by CR-LF will terminate the list.

Date and time are in the same format as the NEWGROUPS command.

A newsgroup name containing a "*" (an asterisk) may be specified to broaden the article search to some or all newsgroups. The asterisk will be extended to match any part of a newsgroup name (e.g., net.micro* will match net.micro.wombat, net.micro.apple, etc). Thus if only an asterisk is given as the newsgroup name, all newsgroups will be searched for new news.

(Please note that the asterisk "*" expansion is a general replacement; in particular, the specification of e.g., net.*.unix should be correctly expanded to embrace names such as net.wombat.unix and net.whocares.unix.)

Conversely, if no asterisk appears in a given newsgroup name, only the specified newsgroup will be searched for new articles. Newsgroup names must be chosen from those returned in the listing of available groups. Multiple newsgroup names (including a "*") may be specified in this command, separated by a comma. No comma shall appear after the last newsgroup in the list. [Implementors are cautioned to keep the 512 character command length limit in mind.]

The exclamation point ("!") may be used to negate a match. This can be used to selectively omit certain newsgroups from an otherwise larger list. For example, a newsgroups specification of "net.*,mod.*,!mod.map.*" would specify that all net.<anything> and all mod.<anything> EXCEPT mod.map.<anything> newsgroup names would be matched. If used, the exclamation point must appear as the first character of the given newsgroup name or pattern.

The optional parameter "distributions" is a list of distribution groups, enclosed in angle brackets. If specified, the distribution portion of an article's newsgroup (e.g, 'net' in 'net.wombat') will be examined for a match with the distribution categories listed, and only those articles which have at least one newsgroup belonging to

the list of distributions will be listed. If more than one distribution group is to be supplied, they must be separated by commas within the angle brackets.

The use of the IHAVE, NEWNEWS, and NEWGROUPS commands to distribute news is discussed in an earlier part of this document.

Please note that an empty list (i.e., the text body returned by this command consists only of the terminating period) is a possible valid response, and indicates that there is currently no new news.

3.8.2. Responses

230 list of new articles by message-id follows

3.9. The NEXT command

3.9.1. NEXT

NEXT

The internally maintained "current article pointer" is advanced to the next article in the current newsgroup. If no more articles remain in the current group, an error message is returned and the current article remains selected.

The internally-maintained "current article pointer" is set by this command.

A response indicating the current article number, and the message-id string will be returned. No text is sent in response to this command.

3.9.2. Responses

223 n a article retrieved - request text separately
(n = article number, a = unique article id)
412 no newsgroup selected
420 no current article has been selected
421 no next article in this group

3.10. The POST command

3.10.1. POST

POST

If posting is allowed, response code 340 is returned to indicate that the article to be posted should be sent. Response code 440 indicates that posting is prohibited for some installation-dependent reason.

If posting is permitted, the article should be presented in the format specified by RFC850, and should include all required header lines. After the article's header and body have been completely sent by the client to the server, a further response code will be returned to indicate success or failure of the posting attempt.

The text forming the header and body of the message to be posted should be sent by the client using the conventions for text received from the news server: A single period (".") on a line indicates the end of the text, with lines starting with a period in the original text having that period doubled during transmission.

No attempt shall be made by the server to filter characters, fold or limit lines, or otherwise process incoming text. It is our intent that the server just pass the incoming message to be posted to the server installation's news posting software, which is separate from this specification. See RFC850 for more details.

Since most installations will want the client news program to allow the user to prepare his message using some sort of text editor, and transmit it to the server for posting only after it is composed, the client program should take note of the herald message that greeted it when the connection was first established. This message indicates whether postings from that client are permitted or not, and can be used to caution the user that his access is read-only if that is the case. This will prevent the user from wasting a good deal of time composing a message only to find posting of the message was denied. The method and determination of which clients and hosts may post is installation dependent and is not covered by this specification.

3.10.2. Responses

240 article posted ok
340 send article to be posted. End with <CR-LF>.<CR-LF>
440 posting not allowed
441 posting failed

(for reference, one of the following codes will be sent upon initial connection; the client program should determine whether posting is generally permitted from these:) 200 server ready - posting allowed
201 server ready - no posting allowed

3.11. The QUIT command

3.11.1. QUIT

QUIT

The server process acknowledges the QUIT command and then closes the connection to the client. This is the preferred method for a client to indicate that it has finished all its transactions with the NNTP server.

If a client simply disconnects (or the connection times out, or some other fault occurs), the server should gracefully cease its attempts to service the client.

3.11.2. Responses

205 closing connection - goodbye!

3.12. The SLAVE command

3.12.1. SLAVE

SLAVE

Indicates to the server that this client connection is to a slave server, rather than a user.

This command is intended for use in separating connections to single users from those to subsidiary ("slave") servers. It may be used to indicate that priority should therefore be given to requests from this client, as it is presumably serving more than one person. It might also be used to determine which connections to close when system load levels are exceeded, perhaps giving preference to slave servers. The actual use this command is put to is entirely implementation dependent, and may vary from one host to another. In NNTP servers which do not give priority to slave servers, this command must nonetheless be recognized and acknowledged.

3.12.2. Responses

202 slave status noted

4. Sample Conversations

These are samples of the conversations that might be expected with the news server in hypothetical sessions. The notation C: indicates commands sent to the news server from the client program; S: indicate responses received from the server by the client.

4.1. Example 1 - relative access with NEXT

```
S:      (listens at TCP port 119)

C:      (requests connection on TCP port 119)
S:      200 wombatvax news server ready - posting ok

(client asks for a current newsgroup list)
C:      LIST
S:      215 list of newsgroups follows
S:      net.wombats 00543 00501 y
S:      net.unix-wizards 10125 10011 y
        (more information here)
S:      net.idiots 00100 00001 n
S:      .

(client selects a newsgroup)
C:      GROUP net.unix-wizards
S:      211 104 10011 10125 net.unix-wizards group selected
        (there are 104 articles on file, from 10011 to 10125)

(client selects an article to read)
C:      STAT 10110
S:      223 10110 <23445@sdcsvax.ARPA> article retrieved - statistics
        only (article 10110 selected, its message-id is
        <23445@sdcsvax.ARPA>)

(client examines the header)
C:      HEAD
S:      221 10110 <23445@sdcsvax.ARPA> article retrieved - head
        follows (text of the header appears here)
S:      .

(client wants to see the text body of the article)
C:      BODY
S:      222 10110 <23445@sdcsvax.ARPA> article retrieved - body
        follows (body text here)
S:      .

(client selects next article in group)
```

```
   C:      NEXT
   S:      223 10113 <21495@nudebch.uucp> article retrieved - statistics
           only (article 10113 was next in group)

   (client finishes session)
   C:      QUIT
   S:      205 goodbye.
```

4.2. Example 2 - absolute article access with ARTICLE

```
   S:      (listens at TCP port 119)

   C:      (requests connection on TCP port 119)
   S:      201 UCB-VAX netnews server ready -- no posting allowed

   C:      GROUP msgs
   S:      211 103 402 504 msgs Your new group is msgs
           (there are 103 articles, from 402 to 504)

   C:      ARTICLE 401
   S:      423 No such article in this newsgroup

   C:      ARTICLE 402
   S:      220 402 <4105@ucbvax.ARPA> Article retrieved, text follows
   S:      (article header and body follow)
   S:      .

   C:      HEAD 403
   S:      221 403 <3108@mcvax.UUCP> Article retrieved, header follows
   S:      (article header follows)
   S:      .

   C:      QUIT
   S:      205 UCB-VAX news server closing connection.  Goodbye.
```

4.3. Example 3 - NEWGROUPS command

```
   S:      (listens at TCP port 119)

   C:      (requests connection on TCP port 119)
   S:      200 Imaginary Institute News Server ready (posting ok)

   (client asks for new newsgroups since April 3, 1985)
   C:      NEWGROUPS 850403 020000

   S:      231 New newsgroups since 03/04/85 02:00:00 follow
   S:      net.music.gdead
   S:      net.games.sources
   S:      .

   C:      GROUP net.music.gdead
   S:      211 0 1 1 net.music.gdead Newsgroup selected
           (there are no articles in that newsgroup, and
           the first and last article numbers should be ignored)

   C:      QUIT
   S:      205 Imaginary Institute news server ceasing service.  Bye!
```

4.4. Example 4 - posting a news article

```
   S:      (listens at TCP port 119)

   C:      (requests connection on TCP port 119)
   S:      200 BANZAIVAX news server ready, posting allowed.

   C:      POST
   S:      340 Continue posting; Period on a line by itself to end
   C:      (transmits news article in RFC850 format)
   C:      .
   S:      240 Article posted successfully.

   C:      QUIT
   S:      205 BANZAIVAX closing connection.  Goodbye.
```

4.5. Example 5 - interruption due to operator request

```
   S:      (listens at TCP port 119)

   C:      (requests connection on TCP port 119)
   S:      201 genericvax news server ready, no posting allowed.

           (assume normal conversation for some time, and
           that a newsgroup has been selected)

   C:      NEXT
   S:      223 1013 <5734@mcvax.UUCP> Article retrieved; text separate.

   C:      HEAD
   C:      221 1013 <5734@mcvax.UUCP> Article retrieved; head follows.

   S:      (sends head of article, but halfway through is
           interrupted by an operator request.  The following
           then occurs, without client intervention.)
```

```
   S:      (ends current line with a CR-LF pair)
   S:      .
   S:      400 Connection closed by operator.  Goodbye.
   S:      (closes connection)
```

4.6. Example 6 - Using the news server to distribute news between systems.

```
   S:      (listens at TCP port 119)

   C:      (requests connection on TCP port 119)
   S:      201 Foobar NNTP server ready (no posting)

   (client asks for new newsgroups since 2 am, May 15, 1985)
   C:      NEWGROUPS 850515 020000
   S:      235 New newsgroups since 850515 follow
   S:      net.fluff
   S:      net.lint
   S:      .

   (client asks for new news articles since 2 am, May 15, 1985)
   C:      NEWNEWS * 850515 020000
   S:      230 New news since 850515 020000 follows
   S:      <1772@foo.UUCP>
   S:      <87623@baz.UUCP>
   S:      <17872@GOLD.CSNET>
   S:      .

   (client asks for article <1772@foo.UUCP>)
   C:      ARTICLE <1772@foo.UUCP>
   S:      220 <1772@foo.UUCP> All of article follows
   S:      (sends entire message)
   S:      .

   (client asks for article <87623@baz.UUCP>
   C:      ARTICLE <87623@baz.UUCP>
   S:      220 <87623@baz.UUCP> All of article follows
   S:      (sends entire message)
   S:      .

   (client asks for article <17872@GOLD.CSNET>
   C:      ARTICLE <17872@GOLD.CSNET>
   S:      220 <17872@GOLD.CSNET> All of article follows
   S:      (sends entire message)
   S:      .
```



```
   (client offers an article it has received recently)
   C:      IHAVE <4105@ucbvax.ARPA>
   S:      435 Already seen that one, where you been?

   (client offers another article)
   C:      IHAVE <4106@ucbvax.ARPA>
   S:      335 News to me!  <CRLF.CRLF> to end.
   C:      (sends article)
   C:      .
   S:      235 Article transferred successfully.  Thanks.

   (or)

   S:      436 Transfer failed.

   (client is all through with the session)
   C:      QUIT
   S:      205 Foobar NNTP server bids you farewell.
```

4.7. Summary of commands and responses.

The following are the commands recognized and responses returned by the NNTP server.

4.7.1. Commands

```
   ARTICLE
   BODY
   GROUP
   HEAD
   HELP
   IHAVE
   LAST
   LIST
   NEWGROUPS
   NEWNEWS
   NEXT
   POST
   QUIT
   SLAVE
   STAT
```

4.7.2. Responses

```
   100 help text follows
   199 debug output
```

```
200 server ready - posting allowed
201 server ready - no posting allowed
202 slave status noted
205 closing connection - goodbye!
211 n f l s group selected
215 list of newsgroups follows
220 n <a> article retrieved - head and body follow 221 n <a> article
retrieved - head follows
222 n <a> article retrieved - body follows
223 n <a> article retrieved - request text separately 230 list of new
articles by message-id follows
231 list of new newsgroups follows
235 article transferred ok
240 article posted ok

335 send article to be transferred.  End with <CR-LF>.<CR-LF>
340 send article to be posted. End with <CR-LF>.<CR-LF>

400 service discontinued
411 no such news group
412 no newsgroup has been selected
420 no current article has been selected
421 no next article in this group
422 no previous article in this group
423 no such article number in this group
430 no such article found
435 article not wanted - do not send it
436 transfer failed - try again later
437 article rejected - do not try again.
440 posting not allowed
441 posting failed

500 command not recognized
501 command syntax error
502 access restriction or permission denied
503 program fault - command not performed
```

4.8. A Brief Word about the USENET News System

In the UNIX world, which traditionally has been linked by 1200 baud dial-up telephone lines, the USENET News system has evolved to handle central storage, indexing, retrieval, and distribution of news. With the exception of its underlying transport mechanism (UUCP), USENET News is an efficient means of providing news and bulletin service to subscribers on UNIX and other hosts worldwide. The USENET News

system is discussed in detail in RFC 850. It runs on most versions of UNIX and on many other operating systems, and is customarily distributed without charge.

USENET uses a spooling area on the UNIX host to store news articles, one per file. Each article consists of a series of heading text, which contain the sender's identification and organizational affiliation, timestamps, electronic mail reply paths, subject, newsgroup (subject category), and the like. A complete news article is reproduced in its entirety below. Please consult RFC 850 for more details.

```
Relay-Version: version B 2.10.3 4.3bsd-beta 6/6/85; site
sdcsvax.UUCP
Posting-Version: version B 2.10.1 6/24/83 SMI; site unitek.uucp
Path:sdcsvax!sdcrdcf!hplabs!qantel!ihnp4!alberta!ubc-vision!unitek
!honman
From: honman@unitek.uucp (Man Wong)
Newsgroups: net.unix-wizards
Subject: foreground -> background ?
Message-ID: <167@unitek.uucp>
Date: 25 Sep 85 23:51:52 GMT
Date-Received: 29 Sep 85 09:54:48 GMT
Reply-To: honman@unitek.UUCP (Hon-Man Wong)
Distribution: net.all
Organization: Unitek Technologies Corporation
Lines: 12

I have a process (C program) which generates a child and waits for
it to return.  What I would like to do is to be able to run the
child process interactively for a while before kicking itself into
the background so I can return to the parent process (while the
child process is RUNNING in the background).  Can it be done?  And
if it can, how?

Please reply by E-mail.  Thanks in advance.

Hon-Man Wong
```

5. References

[1] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", RFC-822, Department of Electrical Engineering, University of Delaware, August, 1982.

[2] Horton, M., "Standard for Interchange of USENET Messages", RFC-850, USENET Project, June, 1983.

[3] Postel, J., "Transmission Control Protocol- DARPA Internet Program Protocol Specification", RFC-793, USC/Information Sciences Institute, September, 1981.

[4] Postel, J., "Simple Mail Transfer Protocol", RFC-821, USC/Information Sciences Institute, August, 1982.

6. Acknowledgements

The authors wish to express their heartfelt thanks to those many people who contributed to this specification, and especially to Erik Fair and Chuq von Rospach, without whose inspiration this whole thing would not have been necessary.

7. Notes

<1> UNIX is a trademark of Bell Laboratories.

Network Working Group
Request for Comments: 2396
Updates: 1808, 1738
Category: Standards Track

T. Berners-Lee
MIT/LCS
R. Fielding
U.C. Irvine
L. Masinter
Xerox Corporation
August 1998

Uniform Resource Identifiers (URI): Generic Syntax

Status of this Memo

Copyright Notice

IESG Note

This paper describes a "superset" of operations that can be applied to URI. It consists of both a grammar and a description of basic functionality for URI. To understand what is a valid URI, both the grammar and the associated description have to be studied. Some of the functionality described is not applicable to all URI schemes, and some operations are only possible when certain media types are retrieved using the URI, regardless of the scheme used.

Abstract

A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource. This document defines the generic syntax of URI, including both absolute and relative forms, and guidelines for their use; it revises and replaces the generic definitions in RFC 1738 and RFC 1808.

This document defines a grammar that is a superset of all valid URI, such that an implementation can parse the common components of a URI reference without knowing the scheme-specific requirements of every possible identifier type. This document does not define a generative grammar for URI; that task will be performed by the individual specifications of each URI scheme.

1. Introduction

Uniform Resource Identifiers (URI) provide a simple and extensible means for identifying a resource. This specification of URI syntax and semantics is derived from concepts introduced by the World Wide Web global information initiative, whose use of such objects dates from 1990 and is described in "Universal Resource Identifiers in WWW" [RFC1630]. The specification of URI is designed to meet the recommendations laid out in "Functional Recommendations for Internet Resource Locators" [RFC1736] and "Functional Requirements for Uniform Resource Names" [RFC1737].

This document updates and merges "Uniform Resource Locators" [RFC1738] and "Relative Uniform Resource Locators" [RFC1808] in order to define a single, generic syntax for all URI. It excludes those portions of RFC 1738 that defined the specific syntax of individual URL schemes; those portions will be updated as separate documents, as will the process for registration of new URI schemes. This document does not discuss the issues and recommendation for dealing with characters outside of the US-ASCII character set [ASCII]; those recommendations are discussed in a separate document.

All significant changes from the prior RFCs are noted in Appendix G.

1.1 Overview of URI

URI are characterized by the following definitions:

Uniform
: Uniformity provides several benefits: it allows different types of resource identifiers to be used in the same context, even when the mechanisms used to access those resources may differ; it allows uniform semantic interpretation of common syntactic conventions across different types of resource identifiers; it allows introduction of new types of resource identifiers without interfering with the way that existing identifiers are used; and, it allows the identifiers to be reused in many different contexts, thus permitting new applications or protocols to leverage a pre-existing, large, and widely-used set of resource identifiers.

Resource
: A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources.

The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content---the entities to which it currently corresponds---changes over time, provided that the conceptual mapping is not changed in the process.

Identifier
An identifier is an object that can act as a reference to something that has identity. In the case of URI, the object is a sequence of characters with a restricted syntax.

Having identified a resource, a system may perform a variety of operations on the resource, as might be characterized by such words as `access', `update', `replace', or `find attributes'.

1.2. URI, URL, and URN

A URI can be further classified as a locator, a name, or both. The term "Uniform Resource Locator" (URL) refers to the subset of URI that identify resources via a representation of their primary access mechanism (e.g., their network "location"), rather than identifying the resource by name or by some other attribute(s) of that resource. The term "Uniform Resource Name" (URN) refers to the subset of URI that are required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.

The URI scheme (Section 3.1) defines the namespace of the URI, and thus may further restrict the syntax and semantics of identifiers using that scheme. This specification defines those elements of the URI syntax that are either required of all URI schemes or are common to many URI schemes. It thus defines the syntax and semantics that are needed to implement a scheme-independent parsing mechanism for URI references, such that the scheme-dependent handling of a URI can be postponed until the scheme-dependent semantics are needed. We use the term URL below when describing syntax or semantics that only apply to locators.

Although many URL schemes are named after protocols, this does not imply that the only way to access the URL's resource is via the named protocol. Gateways, proxies, caches, and name resolution services might be used to access some resources, independent of the protocol of their origin, and the resolution of some URL may require the use of more than one protocol (e.g., both DNS and HTTP are typically used to access an "http" URL's resource when it can't be found in a local cache).

A URN differs from a URL in that it's primary purpose is persistent labeling of a resource with an identifier. That identifier is drawn from one of a set of defined namespaces, each of which has its own set name structure and assignment procedures. The "urn" scheme has been reserved to establish the requirements for a standardized URN namespace, as defined in "URN Syntax" [RFC2141] and its related specifications.

Most of the examples in this specification demonstrate URL, since they allow the most varied use of the syntax and often have a hierarchical namespace. A parser of the URI syntax is capable of parsing both URL and URN references as a generic URI; once the scheme is determined, the scheme-specific parsing can be performed on the generic URI components. In other words, the URI syntax is a superset of the syntax of all URI schemes.

1.3. Example URI

The following examples illustrate URI that are in common use.

ftp://ftp.is.co.za/rfc/rfc1808.txt
-- ftp scheme for File Transfer Protocol services

gopher://spinaltap.micro.umn.edu/00/Weather/California/Los%20Angeles
-- gopher scheme for Gopher and Gopher+ Protocol services

http://www.math.uio.no/faq/compression-faq/part1.html
-- http scheme for Hypertext Transfer Protocol services

mailto:mduerst@ifi.unizh.ch
-- mailto scheme for electronic mail addresses

news:comp.infosystems.www.servers.unix
-- news scheme for USENET news groups and articles

telnet://melvyl.ucop.edu/
-- telnet scheme for interactive services via the TELNET Protocol

1.4. Hierarchical URI and Relative Forms

An absolute identifier refers to a resource independent of the context in which the identifier is used. In contrast, a relative identifier refers to a resource by describing the difference within a hierarchical namespace between the current context and an absolute identifier of the resource.

Some URI schemes support a hierarchical naming system, where the hierarchy of the name is denoted by a "/" delimiter separating the components in the scheme. This document defines a scheme-independent `relative' form of URI reference that can be used in conjunction with a `base' URI (of a hierarchical scheme) to produce another URI. The syntax of hierarchical URI is described in Section 3; the relative URI calculation is described in Section 5.

1.5. URI Transcribability

The URI syntax was designed with global transcribability as one of its main concerns. A URI is a sequence of characters from a very limited set, i.e. the letters of the basic Latin alphabet, digits, and a few special characters. A URI may be represented in a variety of ways: e.g., ink on paper, pixels on a screen, or a sequence of octets in a coded character set. The interpretation of a URI depends only on the characters used and not how those characters are represented in a network protocol.

The goal of transcribability can be described by a simple scenario. Imagine two colleagues, Sam and Kim, sitting in a pub at an international conference and exchanging research ideas. Sam asks Kim for a location to get more information, so Kim writes the URI for the research site on a napkin. Upon returning home, Sam takes out the napkin and types the URI into a computer, which then retrieves the information to which Kim referred.

There are several design concerns revealed by the scenario:

- o A URI is a sequence of characters, which is not always represented as a sequence of octets.

- o A URI may be transcribed from a non-network source, and thus should consist of characters that are most likely to be able to be typed into a computer, within the constraints imposed by keyboards (and related input devices) across languages and locales.

- o A URI often needs to be remembered by people, and it is easier for people to remember a URI when it consists of meaningful components.

These design concerns are not always in alignment. For example, it is often the case that the most meaningful name for a URI component would require characters that cannot be typed into some systems. The ability to transcribe the resource identifier from one medium to another was considered more important than having its URI consist of the most meaningful of components. In local and regional contexts

and with improving technology, users might benefit from being able to use a wider range of characters; such use is not defined in this document.

1.6. Syntax Notation and Common Elements

This document uses two conventions to describe and define the syntax for URI. The first, called the layout form, is a general description of the order of components and component separators, as in

```
<first>/<second>;<third>?<fourth>
```

The component names are enclosed in angle-brackets and any characters outside angle-brackets are literal separators. Whitespace should be ignored. These descriptions are used informally and do not define the syntax requirements.

The second convention is a BNF-like grammar, used to define the formal URI syntax. The grammar is that of [RFC822], except that "|" is used to designate alternatives. Briefly, rules are separated from definitions by an equal "=", indentation is used to continue a rule definition over more than one line, literals are quoted with "", parentheses "(" and ")" are used to group elements, optional elements are enclosed in "[" and "]" brackets, and elements may be preceded with <n>* to designate n or more repetitions of the following element; n defaults to 0.

Unlike many specifications that use a BNF-like grammar to define the bytes (octets) allowed by a protocol, the URI grammar is defined in terms of characters. Each literal in the grammar corresponds to the character it represents, rather than to the octet encoding of that character in any particular coded character set. How a URI is represented in terms of bits and bytes on the wire is dependent upon the character encoding of the protocol used to transport it, or the charset of the document which contains it.

The following definitions are common to many elements:

```
alpha    = lowalpha | upalpha

lowalpha = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
           "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
           "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"

upalpha  = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
           "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
           "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
```

```
digit    = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
           "8" | "9"

alphanum = alpha | digit
```

The complete URI syntax is collected in Appendix A.

2. URI Characters and Escape Sequences

URI consist of a restricted set of characters, primarily chosen to aid transcribability and usability both in computer systems and in non-computer communications. Characters used conventionally as delimiters around URI were excluded. The restricted set of characters consists of digits, letters, and a few graphic symbols were chosen from those common to most of the character encodings and input facilities available to Internet users.

```
uric          = reserved | unreserved | escaped
```

Within a URI, characters are either used as delimiters, or to represent strings of data (octets) within the delimited portions. Octets are either represented directly by a character (using the US-ASCII character for that octet [ASCII]) or by an escape encoding. This representation is elaborated below.

2.1 URI and non-ASCII characters

The relationship between URI and characters has been a source of confusion for characters that are not part of US-ASCII. To describe the relationship, it is useful to distinguish between a "character" (as a distinguishable semantic entity) and an "octet" (an 8-bit byte). There are two mappings, one from URI characters to octets, and a second from octets to original characters:

URI character sequence->octet sequence->original character sequence

A URI is represented as a sequence of characters, not as a sequence of octets. That is because URI might be "transported" by means that are not through a computer network, e.g., printed on paper, read over the radio, etc.

A URI scheme may define a mapping from URI characters to octets; whether this is done depends on the scheme. Commonly, within a delimited component of a URI, a sequence of characters may be used to represent a sequence of octets. For example, the character "a" represents the octet 97 (decimal), while the character sequence "%", "0", "a" represents the octet 10 (decimal).

There is a second translation for some resources: the sequence of octets defined by a component of the URI is subsequently used to represent a sequence of characters. A 'charset' defines this mapping. There are many charsets in use in Internet protocols. For example, UTF-8 [UTF-8] defines a mapping from sequences of octets to sequences of characters in the repertoire of ISO 10646.

In the simplest case, the original character sequence contains only characters that are defined in US-ASCII, and the two levels of mapping are simple and easily invertible: each 'original character' is represented as the octet for the US-ASCII code for it, which is, in turn, represented as either the US-ASCII character, or else the "%" escape sequence for that octet.

For original character sequences that contain non-ASCII characters, however, the situation is more difficult. Internet protocols that transmit octet sequences intended to represent character sequences are expected to provide some way of identifying the charset used, if there might be more than one [RFC2277]. However, there is currently no provision within the generic URI syntax to accomplish this identification. An individual URI scheme may require a single charset, define a default charset, or provide a way to indicate the charset used.

It is expected that a systematic treatment of character encoding within URI will be developed as a future modification of this specification.

2.2. Reserved Characters

Many URI include components consisting of or delimited by, certain special characters. These characters are called "reserved", since their usage within the URI component is limited to their reserved purpose. If the data for a URI component would conflict with the reserved purpose, then the conflicting data must be escaped before forming the URI.

```
reserved    = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
              "$" | ","
```

The "reserved" syntax class above refers to those characters that are allowed within a URI, but which may not be allowed within a particular component of the generic URI syntax; they are used as delimiters of the components described in Section 3.

Characters in the "reserved" set are not reserved in all contexts. The set of characters actually reserved within any given URI component is defined by that component. In general, a character is reserved if the semantics of the URI changes if the character is replaced with its escaped US-ASCII encoding.

2.3. Unreserved Characters

Data characters that are allowed in a URI but do not have a reserved purpose are called unreserved. These include upper and lower case letters, decimal digits, and a limited set of punctuation marks and symbols.

```
unreserved  = alphanum | mark

mark        = "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
```

Unreserved characters can be escaped without changing the semantics of the URI, but this should not be done unless the URI is being used in a context that does not allow the unescaped character to appear.

2.4. Escape Sequences

Data must be escaped if it does not have a representation using an unreserved character; this includes data that does not correspond to a printable character of the US-ASCII coded character set, or that corresponds to any US-ASCII character that is disallowed, as explained below.

2.4.1. Escaped Encoding

An escaped octet is encoded as a character triplet, consisting of the percent character "%" followed by the two hexadecimal digits representing the octet code. For example, "%20" is the escaped encoding for the US-ASCII space character.

```
escaped     = "%" hex hex
hex         = digit | "A" | "B" | "C" | "D" | "E" | "F" |
                      "a" | "b" | "c" | "d" | "e" | "f"
```

2.4.2. When to Escape and Unescape

A URI is always in an "escaped" form, since escaping or unescaping a completed URI might change its semantics. Normally, the only time escape encodings can safely be made is when the URI is being created from its component parts; each component may have its own set of characters that are reserved, so only the mechanism responsible for generating or interpreting that component can determine whether or

not escaping a character will change its semantics. Likewise, a URI must be separated into its components before the escaped characters within those components can be safely decoded.

In some cases, data that could be represented by an unreserved character may appear escaped; for example, some of the unreserved "mark" characters are automatically escaped by some systems. If the given URI scheme defines a canonicalization algorithm, then unreserved characters may be unescaped according to that algorithm. For example, "%7e" is sometimes used instead of "~" in an http URL path, but the two are equivalent for an http URL.

Because the percent "%" character always has the reserved purpose of being the escape indicator, it must be escaped as "%25" in order to be used as data within a URI. Implementers should be careful not to escape or unescape the same string more than once, since unescaping an already unescaped string might lead to misinterpreting a percent data character as another escaped character, or vice versa in the case of escaping an already escaped string.

2.4.3. Excluded US-ASCII Characters

Although they are disallowed within the URI syntax, we include here a description of those US-ASCII characters that have been excluded and the reasons for their exclusion.

The control characters in the US-ASCII coded character set are not used within a URI, both because they are non-printable and because they are likely to be misinterpreted by some control mechanisms.

```
control     = <US-ASCII coded characters 00-1F and 7F hexadecimal>
```

The space character is excluded because significant spaces may disappear and insignificant spaces may be introduced when URI are transcribed or typeset or subjected to the treatment of word-processing programs. Whitespace is also used to delimit URI in many contexts.

```
space       = <US-ASCII coded character 20 hexadecimal>
```

The angle-bracket "<" and ">" and double-quote (") characters are excluded because they are often used as the delimiters around URI in text documents and protocol fields. The character "#" is excluded because it is used to delimit a URI from a fragment identifier in URI references (Section 4). The percent character "%" is excluded because it is used for the encoding of escaped characters.

```
delims      = "<" | ">" | "#" | "%" | <">
```

Other characters are excluded because gateways and other transport agents are known to sometimes modify such characters, or they are used as delimiters.

```
unwise      = "{" | "}" | "|" | "\" | "^" | "[" | "]" | "`"
```

Data corresponding to excluded characters must be escaped in order to be properly represented within a URI.

3. URI Syntactic Components

The URI syntax is dependent upon the scheme. In general, absolute URI are written as follows:

```
<scheme>:<scheme-specific-part>
```

An absolute URI contains the name of the scheme being used (<scheme>) followed by a colon (":") and then a string (the <scheme-specific-part>) whose interpretation depends on the scheme.

The URI syntax does not require that the scheme-specific-part have any general structure or set of semantics which is common among all URI. However, a subset of URI do share a common syntax for representing hierarchical relationships within the namespace. This "generic URI" syntax consists of a sequence of four main components:

```
<scheme>://<authority><path>?<query>
```

each of which, except <scheme>, may be absent from a particular URI. For example, some URI schemes do not allow an <authority> component, and others do not use a <query> component.

```
absoluteURI   = scheme ":" ( hier_part | opaque_part )
```

URI that are hierarchical in nature use the slash "/" character for separating hierarchical components. For some file systems, a "/" character (used to denote the hierarchical structure of a URI) is the delimiter used to construct a file name hierarchy, and thus the URI path will look similar to a file pathname. This does NOT imply that the resource is a file or that the URI maps to an actual filesystem pathname.

```
hier_part     = ( net_path | abs_path ) [ "?" query ]

net_path      = "//" authority [ abs_path ]

abs_path      = "/"  path_segments
```


URI that do not make use of the slash "/" character for separating hierarchical components are considered opaque by the generic URI parser.

```
opaque_part   = uric_no_slash *uric

uric_no_slash = unreserved | escaped | ";" | "?" | ":" | "@" |
                "&" | "=" | "+" | "$" | ","
```

We use the term <path> to refer to both the <abs_path> and <opaque_part> constructs, since they are mutually exclusive for any given URI and can be parsed as a single component.

3.1. Scheme Component

Just as there are many different methods of access to resources, there are a variety of schemes for identifying such resources. The URI syntax consists of a sequence of components separated by reserved characters, with the first component defining the semantics for the remainder of the URI string.

Scheme names consist of a sequence of characters beginning with a lower case letter and followed by any combination of lower case letters, digits, plus ("+"), period ("."), or hyphen ("-"). For resiliency, programs interpreting URI should treat upper case letters as equivalent to lower case in scheme names (e.g., allow "HTTP" as well as "http").

```
scheme        = alpha *( alpha | digit | "+" | "-" | "." )
```

Relative URI references are distinguished from absolute URI in that they do not begin with a scheme name. Instead, the scheme is inherited from the base URI, as described in Section 5.2.

3.2. Authority Component

Many URI schemes include a top hierarchical element for a naming authority, such that the namespace defined by the remainder of the URI is governed by that authority. This authority component is typically defined by an Internet-based server or a scheme-specific registry of naming authorities.

```
authority     = server | reg_name
```

The authority component is preceded by a double slash "//" and is terminated by the next slash "/", question-mark "?", or by the end of the URI. Within the authority component, the characters ";", ":", "@", "?", and "/" are reserved.

An authority component is not required for a URI scheme to make use of relative references. A base URI without an authority component implies that any relative reference will also be without an authority component.

3.2.1. Registry-based Naming Authority

The structure of a registry-based naming authority is specific to the URI scheme, but constrained to the allowed characters for an authority component.

```
reg_name      = 1*( unreserved | escaped | "$" | "," |
                    ";" | ":" | "@" | "&" | "=" | "+" )
```

3.2.2. Server-based Naming Authority

URL schemes that involve the direct use of an IP-based protocol to a specified server on the Internet use a common syntax for the server component of the URI's scheme-specific data:

```
<userinfo>@<host>:<port>
```

where <userinfo> may consist of a user name and, optionally, scheme-specific information about how to gain authorization to access the server. The parts "<userinfo>@" and ":<port>" may be omitted.

```
server        = [ [ userinfo "@" ] hostport ]
```

The user information, if present, is followed by a commercial at-sign "@".

```
userinfo      = *( unreserved | escaped |
                   ";" | ":" | "&" | "=" | "+" | "$" | "," )
```

Some URL schemes use the format "user:password" in the userinfo field. This practice is NOT RECOMMENDED, because the passing of authentication information in clear text (such as URI) has proven to be a security risk in almost every case where it has been used.

The host is a domain name of a network host, or its IPv4 address as a set of four decimal digit groups separated by ".". Literal IPv6 addresses are not supported.

```
hostport      = host [ ":" port ]
host          = hostname | IPv4address
hostname      = *( domainlabel "." ) toplabel [ "." ]
domainlabel   = alphanum | alphanum *( alphanum | "-" ) alphanum
toplabel      = alpha | alpha *( alphanum | "-" ) alphanum
```



```
IPv4address   = 1*digit "." 1*digit "." 1*digit "." 1*digit
port          = *digit
```

Hostnames take the form described in Section 3 of [RFC1034] and Section 2.1 of [RFC1123]: a sequence of domain labels separated by ".", each domain label starting and ending with an alphanumeric character and possibly also containing "-" characters. The rightmost domain label of a fully qualified domain name will never start with a digit, thus syntactically distinguishing domain names from IPv4 addresses, and may be followed by a single "." if it is necessary to distinguish between the complete domain name and any local domain. To actually be "Uniform" as a resource locator, a URL hostname should be a fully qualified domain name. In practice, however, the host component may be a local domain literal.

> Note: A suitable representation for including a literal IPv6 address as the host part of a URL is desired, but has not yet been determined or implemented in practice.

The port is the network port number for the server. Most schemes designate protocols that have a default port number. Another port number may optionally be supplied, in decimal, separated from the host by a colon. If the port is omitted, the default port number is assumed.

3.3. Path Component

The path component contains data, specific to the authority (or the scheme if there is no authority component), identifying the resource within the scope of that scheme and authority.

```
path          = [ abs_path | opaque_part ]

path_segments = segment *( "/" segment )
segment       = *pchar *( ";" param )
param         = *pchar

pchar         = unreserved | escaped |
                  ":" | "@" | "&" | "=" | "+" | "$" | ","
```

The path may consist of a sequence of path segments separated by a single slash "/" character. Within a path segment, the characters "/", ";", "=", and "?" are reserved. Each path segment may include a sequence of parameters, indicated by the semicolon ";" character. The parameters are not significant to the parsing of relative references.

3.4. Query Component

The query component is a string of information to be interpreted by the resource.

```
query         = *uric
```

Within a query component, the characters ";", "/", "?", ":", "@", "&", "=", "+", ",", and "$" are reserved.

4. URI References

The term "URI-reference" is used here to denote the common usage of a resource identifier. A URI reference may be absolute or relative, and may have additional information attached in the form of a fragment identifier. However, "the URI" that results from such a reference includes only the absolute URI after the fragment identifier (if any) is removed and after any relative URI is resolved to its absolute form. Although it is possible to limit the discussion of URI syntax and semantics to that of the absolute result, most usage of URI is within general URI references, and it is impossible to obtain the URI from such a reference without also parsing the fragment and resolving the relative form.

```
URI-reference = [ absoluteURI | relativeURI ] [ "#" fragment ]
```

The syntax for relative URI is a shortened form of that for absolute URI, where some prefix of the URI is missing and certain path components ("." and "..") have a special meaning when, and only when, interpreting a relative path. The relative URI syntax is defined in Section 5.

4.1. Fragment Identifier

When a URI reference is used to perform a retrieval action on the identified resource, the optional fragment identifier, separated from the URI by a crosshatch ("#") character, consists of additional reference information to be interpreted by the user agent after the retrieval action has been successfully completed. As such, it is not part of a URI, but is often used in conjunction with a URI.

```
fragment      = *uric
```

The semantics of a fragment identifier is a property of the data resulting from a retrieval action, regardless of the type of URI used in the reference. Therefore, the format and interpretation of fragment identifiers is dependent on the media type [RFC2046] of the retrieval result. The character restrictions described in Section 2

for URI also apply to the fragment in a URI-reference. Individual media types may define additional restrictions or structure within the fragment for specifying different types of "partial views" that can be identified within that media type.

A fragment identifier is only meaningful when a URI reference is intended for retrieval and the result of that retrieval is a document for which the identified fragment is consistently defined.

4.2. Same-document References

A URI reference that does not contain a URI is a reference to the current document. In other words, an empty URI reference within a document is interpreted as a reference to the start of that document, and a reference containing only a fragment identifier is a reference to the identified fragment of that document. Traversal of such a reference should not result in an additional retrieval action. However, if the URI reference occurs in a context that is always intended to result in a new request, as in the case of HTML's FORM element, then an empty URI reference represents the base URI of the current document and should be replaced by that URI when transformed into a request.

4.3. Parsing a URI Reference

A URI reference is typically parsed according to the four main components and fragment identifier in order to determine what components are present and whether the reference is relative or absolute. The individual components are then parsed for their subparts and, if not opaque, to verify their validity.

Although the BNF defines what is allowed in each component, it is ambiguous in terms of differentiating between an authority component and a path component that begins with two slash characters. The greedy algorithm is used for disambiguation: the left-most matching rule soaks up as much of the URI reference string as it is capable of matching. In other words, the authority component wins.

Readers familiar with regular expressions should see Appendix B for a concrete parsing example and test oracle.

5. Relative URI References

It is often the case that a group or "tree" of documents has been constructed to serve a common purpose; the vast majority of URI in these documents point to resources within the tree rather than

outside of it. Similarly, documents located at a particular site are much more likely to refer to other resources at that site than to resources at remote sites.

Relative addressing of URI allows document trees to be partially independent of their location and access scheme. For instance, it is possible for a single set of hypertext documents to be simultaneously accessible and traversable via each of the "file", "http", and "ftp" schemes if the documents refer to each other using relative URI. Furthermore, such document trees can be moved, as a whole, without changing any of the relative references. Experience within the WWW has demonstrated that the ability to perform relative referencing is necessary for the long-term usability of embedded URI.

The syntax for relative URI takes advantage of the <hier_part> syntax of <absoluteURI> (Section 3) in order to express a reference that is relative to the namespace of another hierarchical URI.

```
relativeURI   = ( net_path | abs_path | rel_path ) [ "?" query ]
```

A relative reference beginning with two slash characters is termed a network-path reference, as defined by <net_path> in Section 3. Such references are rarely used.

A relative reference beginning with a single slash character is termed an absolute-path reference, as defined by <abs_path> in Section 3.

A relative reference that does not begin with a scheme name or a slash character is termed a relative-path reference.

```
rel_path      = rel_segment [ abs_path ]

rel_segment   = 1*( unreserved | escaped |
                    ";" | "@" | "&" | "=" | "+" | "$" | "," )
```

Within a relative-path reference, the complete path segments "." and ".." have special meanings: "the current hierarchy level" and "the level above this hierarchy level", respectively. Although this is very similar to their use within Unix-based filesystems to indicate directory levels, these path components are only considered special when resolving a relative-path reference to its absolute form (Section 5.2).

Authors should be aware that a path segment which contains a colon character cannot be used as the first segment of a relative URI path (e.g., "this:that"), because it would be mistaken for a scheme name.

It is therefore necessary to precede such segments with other segments (e.g., "./this:that") in order for them to be referenced as a relative path.

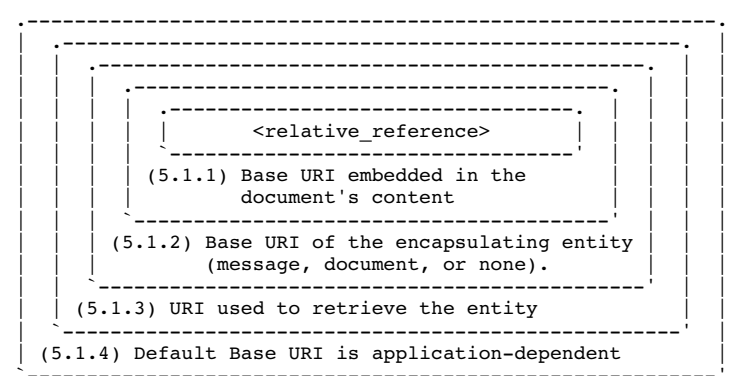
It is not necessary for all URI within a given scheme to be restricted to the <hier_part> syntax, since the hierarchical properties of that syntax are only necessary when relative URI are used within a particular document. Documents can only make use of relative URI when their base URI fits within the <hier_part> syntax. It is assumed that any document which contains a relative reference will also have a base URI that obeys the syntax. In other words, relative URI cannot be used within a document that has an unsuitable base URI.

Some URI schemes do not allow a hierarchical syntax matching the <hier_part> syntax, and thus cannot use relative references.

5.1. Establishing a Base URI

The term "relative URI" implies that there exists some absolute "base URI" against which the relative reference is applied. Indeed, the base URI is necessary to define the semantics of any relative URI reference; without it, a relative reference is meaningless. In order for relative URI to be usable within a document, the base URI of that document must be known to the parser.

The base URI of a document can be established in one of four ways, listed below in order of precedence. The order of precedence can be thought of in terms of layers, where the innermost defined base URI has the highest precedence. This can be visualized graphically as:



5.1.1. Base URI within Document Content

Within certain document media types, the base URI of the document can be embedded within the content itself such that it can be readily obtained by a parser. This can be useful for descriptive documents, such as tables of content, which may be transmitted to others through protocols other than their usual retrieval context (e.g., E-Mail or USENET news).

It is beyond the scope of this document to specify how, for each media type, the base URI can be embedded. It is assumed that user agents manipulating such media types will be able to obtain the appropriate syntax from that media type's specification. An example of how the base URI can be embedded in the Hypertext Markup Language (HTML) [RFC1866] is provided in Appendix D.

A mechanism for embedding the base URI within MIME container types (e.g., the message and multipart types) is defined by MHTML [RFC2110]. Protocols that do not use the MIME message header syntax, but which do allow some form of tagged metainformation to be included within messages, may define their own syntax for defining the base URI as part of a message.

5.1.2. Base URI from the Encapsulating Entity

If no base URI is embedded, the base URI of a document is defined by the document's retrieval context. For a document that is enclosed within another entity (such as a message or another document), the retrieval context is that entity; thus, the default base URI of the document is the base URI of the entity in which the document is encapsulated.

5.1.3. Base URI from the Retrieval URI

If no base URI is embedded and the document is not encapsulated within some other entity (e.g., the top level of a composite entity), then, if a URI was used to retrieve the base document, that URI shall be considered the base URI. Note that if the retrieval was the result of a redirected request, the last URI used (i.e., that which resulted in the actual retrieval of the document) is the base URI.

5.1.4. Default Base URI

If none of the conditions described in Sections 5.1.1--5.1.3 apply, then the base URI is defined by the context of the application. Since this definition is necessarily application-dependent, failing

to define the base URI using one of the other methods may result in the same content being interpreted differently by different types of application.

It is the responsibility of the distributor(s) of a document containing relative URI to ensure that the base URI for that document can be established. It must be emphasized that relative URI cannot be used reliably in situations where the document's base URI is not well-defined.

5.2. Resolving Relative References to Absolute Form

This section describes an example algorithm for resolving URI references that might be relative to a given base URI.

The base URI is established according to the rules of Section 5.1 and parsed into the four main components as described in Section 3. Note that only the scheme component is required to be present in the base URI; the other components may be empty or undefined. A component is undefined if its preceding separator does not appear in the URI reference; the path component is never undefined, though it may be empty. The base URI's query component is not used by the resolution algorithm and may be discarded.

For each URI reference, the following steps are performed in order:

1) The URI reference is parsed into the potential four components and fragment identifier, as described in Section 4.3.

2) If the path component is empty and the scheme, authority, and query components are undefined, then it is a reference to the current document and we are done. Otherwise, the reference URI's query and fragment components are defined as found (or not found) within the URI reference and not inherited from the base URI.

3) If the scheme component is defined, indicating that the reference starts with a scheme name, then the reference is interpreted as an absolute URI and we are done. Otherwise, the reference URI's scheme is inherited from the base URI's scheme component.

   Due to a loophole in prior specifications [RFC1630], some parsers allow the scheme name to be present in a relative URI if it is the same as the base URI scheme. Unfortunately, this can conflict with the correct parsing of non-hierarchical URI. For backwards compatibility, an implementation may work around such references by removing the scheme if it matches that of the base URI and the scheme is known to always use the <hier_part> syntax. The parser

can then continue with the steps below for the remainder of the reference components. Validating parsers should mark such a misformed relative reference as an error.

4) If the authority component is defined, then the reference is a network-path and we skip to step 7. Otherwise, the reference URI's authority is inherited from the base URI's authority component, which will also be undefined if the URI scheme does not use an authority component.

5) If the path component begins with a slash character ("/"), then the reference is an absolute-path and we skip to step 7.

6) If this step is reached, then we are resolving a relative-path reference. The relative path needs to be merged with the base URI's path. Although there are many ways to do this, we will describe a simple method using a separate string buffer.

   a) All but the last segment of the base URI's path component is copied to the buffer. In other words, any characters after the last (right-most) slash character, if any, are excluded.

   b) The reference's path component is appended to the buffer string.

   c) All occurrences of "./", where "." is a complete path segment, are removed from the buffer string.

   d) If the buffer string ends with "." as a complete path segment, that "." is removed.

   e) All occurrences of "/../", where is a complete path segment not equal to "..", are removed from the buffer string. Removal of these path segments is performed iteratively, removing the leftmost matching pattern on each iteration, until no matching pattern remains.

   f) If the buffer string ends with "/..", where is a complete path segment not equal to "..", that "/.." is removed.

   g) If the resulting buffer string still begins with one or more complete path segments of "..", then the reference is considered to be in error. Implementations may handle this error by retaining these components in the resolved path (i.e., treating them as part of the final URI), by removing them from the resolved path (i.e., discarding relative levels above the root), or by avoiding traversal of the reference.

   h) The remaining buffer string is the reference URI's new path component.

7) The resulting URI components, including any inherited from the base URI, are recombined to give the absolute form of the URI reference. Using pseudocode, this would be

```
result = ""

if scheme is defined then
    append scheme to result
    append ":" to result

if authority is defined then
    append "//" to result
    append authority to result

append path to result

if query is defined then
    append "?" to result
    append query to result

if fragment is defined then
    append "#" to result
    append fragment to result

return result
```

   Note that we must be careful to preserve the distinction between a component that is undefined, meaning that its separator was not present in the reference, and a component that is empty, meaning that the separator was present and was immediately followed by the next component separator or the end of the reference.

The above algorithm is intended to provide an example by which the output of implementations can be tested -- implementation of the algorithm itself is not required. For example, some systems may find it more efficient to implement step 6 as a pair of segment stacks being merged, rather than as a series of string pattern replacements.

   Note: Some WWW client applications will fail to separate the reference's query component from its path component before merging the base and reference paths in step 6 above. This may result in a loss of information if the query component contains the strings "/../" or "/./".

Resolution examples are provided in Appendix C.

6. URI Normalization and Equivalence

In many cases, different URI strings may actually identify the identical resource. For example, the host names used in URL are actually case insensitive, and the URL <http://www.XEROX.com> is equivalent to <http://www.xerox.com>. In general, the rules for equivalence and definition of a normal form, if any, are scheme dependent. When a scheme uses elements of the common syntax, it will also use the common syntax equivalence rules, namely that the scheme and hostname are case insensitive and a URL with an explicit ":port", where the port is the default for the scheme, is equivalent to one where the port is elided.

7. Security Considerations

A URI does not in itself pose a security threat. Users should beware that there is no general guarantee that a URL, which at one time located a given resource, will continue to do so. Nor is there any guarantee that a URL will not locate a different resource at some later point in time, due to the lack of any constraint on how a given authority apportions its namespace. Such a guarantee can only be obtained from the person(s) controlling that namespace and the resource in question. A specific URI scheme may include additional semantics, such as name persistence, if those semantics are required of all naming authorities for that scheme.

It is sometimes possible to construct a URL such that an attempt to perform a seemingly harmless, idempotent operation, such as the retrieval of an entity associated with the resource, will in fact cause a possibly damaging remote operation to occur. The unsafe URL is typically constructed by specifying a port number other than that reserved for the network protocol in question. The client unwittingly contacts a site that is in fact running a different protocol. The content of the URL contains instructions that, when interpreted according to this other protocol, cause an unexpected operation. An example has been the use of a gopher URL to cause an unintended or impersonating message to be sent via a SMTP server.

Caution should be used when using any URL that specifies a port number other than the default for the protocol, especially when it is a number within the reserved space.

Care should be taken when a URL contains escaped delimiters for a given protocol (for example, CR and LF characters for telnet protocols) that these are not unescaped before transmission. This might violate the protocol, but avoids the potential for such

characters to be used to simulate an extra operation or parameter in that protocol, which might lead to an unexpected and possibly harmful remote operation to be performed.

It is clearly unwise to use a URL that contains a password which is intended to be secret. In particular, the use of a password within the 'userinfo' component of a URL is strongly disrecommended except in those rare cases where the 'password' parameter is intended to be public.

8. Acknowledgements

This document was derived from RFC 1738 [RFC1738] and RFC 1808 [RFC1808]; the acknowledgements in those specifications still apply. In addition, contributions by Gisle Aas, Martin Beet, Martin Duerst, Jim Gettys, Martijn Koster, Dave Kristol, Daniel LaLiberte, Foteos Macrides, James Marshall, Ryan Moats, Keith Moore, and Lauren Wood are gratefully acknowledged.

9. References

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[RFC1630] Berners-Lee, T., "Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web", RFC 1630, June 1994.

[RFC1738] Berners-Lee, T., Masinter, L., and M. McCahill, Editors, "Uniform Resource Locators (URL)", RFC 1738, December 1994.

[RFC1866] Berners-Lee T., and D. Connolly, "HyperText Markup Language Specification -- 2.0", RFC 1866, November 1995.

[RFC1123] Braden, R., Editor, "Requirements for Internet Hosts -- Application and Support", STD 3, RFC 1123, October 1989.

[RFC822] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, August 1982.

[RFC1808] Fielding, R., "Relative Uniform Resource Locators", RFC 1808, June 1995.

[RFC2046] Freed, N., and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC1736] Kunze, J., "Functional Recommendations for Internet Resource Locators", RFC 1736, February 1995.

[RFC2141] Moats, R., "URN Syntax", RFC 2141, May 1997.

[RFC1034] Mockapetris, P., "Domain Names - Concepts and Facilities", STD 13, RFC 1034, November 1987.

[RFC2110] Palme, J., and A. Hopmann, "MIME E-mail Encapsulation of Aggregate Documents, such as HTML (MHTML)", RFC 2110, March 1997.

[RFC1737] Sollins, K., and L. Masinter, "Functional Requirements for Uniform Resource Names", RFC 1737, December 1994.

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

[UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

10. Authors' Addresses

Tim Berners-Lee
World Wide Web Consortium
MIT Laboratory for Computer Science, NE43-356
545 Technology Square
Cambridge, MA 02139

Fax: +1(617)258-8682
EMail: timbl@w3.org

Roy T. Fielding
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425

Fax: +1(949)824-1715
EMail: fielding@ics.uci.edu

Larry Masinter
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94034

Fax: +1(415)812-4333
EMail: masinter@parc.xerox.com

A. Collected BNF for URI

```
URI-reference = [ absoluteURI | relativeURI ] [ "#" fragment ]
absoluteURI   = scheme ":" ( hier_part | opaque_part )
relativeURI   = ( net_path | abs_path | rel_path ) [ "?" query ]

hier_part     = ( net_path | abs_path ) [ "?" query ]
opaque_part   = uric_no_slash *uric

uric_no_slash = unreserved | escaped | ";" | "?" | ":" | "@" |
                "&" | "=" | "+" | "$" | ","

net_path      = "//" authority [ abs_path ]
abs_path      = "/"  path_segments
rel_path      = rel_segment [ abs_path ]

rel_segment   = 1*( unreserved | escaped |
                    ";" | "@" | "&" | "=" | "+" | "$" | "," )

scheme        = alpha *( alpha | digit | "+" | "-" | "." )

authority     = server | reg_name

reg_name      = 1*( unreserved | escaped | "$" | "," |
                    ";" | ":" | "@" | "&" | "=" | "+" )

server        = [ [ userinfo "@" ] hostport ]
userinfo      = *( unreserved | escaped |
                   ";" | ":" | "&" | "=" | "+" | "$" | "," )

hostport      = host [ ":" port ]
host          = hostname | IPv4address
hostname      = *( domainlabel "." ) toplabel [ "." ]
domainlabel   = alphanum | alphanum *( alphanum | "-" ) alphanum
toplabel      = alpha | alpha *( alphanum | "-" ) alphanum
IPv4address   = 1*digit "." 1*digit "." 1*digit "." 1*digit
port          = *digit

path          = [ abs_path | opaque_part ]
path_segments = segment *( "/" segment )
segment       = *pchar *( ";" param )
param         = *pchar
pchar         = unreserved | escaped |
                ":" | "@" | "&" | "=" | "+" | "$" | ","

query         = *uric

fragment      = *uric
```



```
uric          = reserved | unreserved | escaped
reserved      = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" |
                "$" | ","
unreserved    = alphanum | mark
mark          = "-" | "_" | "." | "!" | "~" | "*" | "'" |
                "(" | ")"

escaped       = "%" hex hex
hex           = digit | "A" | "B" | "C" | "D" | "E" | "F" |
                        "a" | "b" | "c" | "d" | "e" | "f"

alphanum      = alpha | digit
alpha         = lowalpha | upalpha

lowalpha = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
           "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
           "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
upalpha  = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
           "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
           "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
digit    = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
           "8" | "9"
```

B. Parsing a URI Reference with a Regular Expression

As described in Section 4.3, the generic URI syntax is not sufficient to disambiguate the components of some forms of URI. Since the "greedy algorithm" described in that section is identical to the disambiguation method used by POSIX regular expressions, it is natural and commonplace to use a regular expression for parsing the potential four components and fragment identifier of a URI reference.

The following line is the regular expression for breaking-down a URI reference into its components.

```
^(([^:/?#]+):)?(//([^/?#]*))?([^?#]*)(\?([^#]*))?(#(.*))?
 12            3  4          5       6  7        8 9
```

The numbers in the second line above are only to assist readability; they indicate the reference points for each subexpression (i.e., each paired parenthesis). We refer to the value matched for subexpression <n> as $<n>. For example, matching the above expression to

```
http://www.ics.uci.edu/pub/ietf/uri/#Related
```

results in the following subexpression matches:

```
$1 = http:
$2 = http
$3 = //www.ics.uci.edu
$4 = www.ics.uci.edu
$5 = /pub/ietf/uri/
$6 = <undefined>
$7 = <undefined>
$8 = #Related
$9 = Related
```

where <undefined> indicates that the component is not present, as is the case for the query component in the above example. Therefore, we can determine the value of the four components and fragment as

```
scheme    = $2
authority = $4
path      = $5
query     = $7
fragment  = $9
```

and, going in the opposite direction, we can recreate a URI reference from its components using the algorithm in step 7 of Section 5.2.

C. Examples of Resolving Relative URI References

Within an object with a well-defined base URI of

```
http://a/b/c/d;p?q
```

the relative URI would be resolved as follows:

C.1. Normal Examples

```
g:h           =  g:h
g             =  http://a/b/c/g
./g           =  http://a/b/c/g
g/            =  http://a/b/c/g/
/g            =  http://a/g
//g           =  http://g
?y            =  http://a/b/c/?y
g?y           =  http://a/b/c/g?y
#s            =  (current document)#s
g#s           =  http://a/b/c/g#s
g?y#s         =  http://a/b/c/g?y#s
;x            =  http://a/b/c/;x
g;x           =  http://a/b/c/g;x
g;x?y#s       =  http://a/b/c/g;x?y#s
.             =  http://a/b/c/
./            =  http://a/b/c/
..            =  http://a/b/
../           =  http://a/b/
../g          =  http://a/b/g
../..         =  http://a/
../../        =  http://a/
../../g       =  http://a/g
```

C.2. Abnormal Examples

Although the following abnormal examples are unlikely to occur in normal practice, all URI parsers should be capable of resolving them consistently. Each example uses the same base as above.

An empty reference refers to the start of the current document.

```
<>            =  (current document)
```

Parsers must be careful in handling the case where there are more relative path ".." segments than there are hierarchical levels in the base URI's path. Note that the ".." syntax cannot be used to change the authority component of a URI.

```
../../../g    =  http://a/../g
../../../../g =  http://a/../../g
```

In practice, some implementations strip leading relative symbolic elements (".", "..") after applying a relative URI calculation, based on the theory that compensating for obvious author errors is better than allowing the request to fail. Thus, the above two references will be interpreted as "http://a/g" by some implementations.

Similarly, parsers must avoid treating "." and ".." as special when they are not complete components of a relative path.

```
/./g          =  http://a/./g
/../g         =  http://a/../g
g.            =  http://a/b/c/g.
.g            =  http://a/b/c/.g
g..           =  http://a/b/c/g..
..g           =  http://a/b/c/..g
```

Less likely are cases where the relative URI uses unnecessary or nonsensical forms of the "." and ".." complete path segments.

```
./../g        =  http://a/b/g
./g/.         =  http://a/b/c/g/
g/./h         =  http://a/b/c/g/h
g/../h        =  http://a/b/c/h
g;x=1/./y     =  http://a/b/c/g;x=1/y
g;x=1/../y    =  http://a/b/c/y
```

All client applications remove the query component from the base URI before resolving relative URI. However, some applications fail to separate the reference's query and/or fragment components from a relative path before merging it with the base path. This error is rarely noticed, since typical usage of a fragment never includes the hierarchy ("/") character, and the query component is not normally used within relative references.

```
g?y/./x       =  http://a/b/c/g?y/./x
g?y/../x      =  http://a/b/c/g?y/../x
g#s/./x       =  http://a/b/c/g#s/./x
g#s/../x      =  http://a/b/c/g#s/../x
```


Some parsers allow the scheme name to be present in a relative URI if it is the same as the base URI scheme. This is considered to be a loophole in prior specifications of partial URI [RFC1630]. Its use should be avoided.

```
http:g        =  http:g           ; for validating parsers
              |  http://a/b/c/g   ; for backwards compatibility
```

D. Embedding the Base URI in HTML documents

It is useful to consider an example of how the base URI of a document can be embedded within the document's content. In this appendix, we describe how documents written in the Hypertext Markup Language (HTML) [RFC1866] can include an embedded base URI. This appendix does not form a part of the URI specification and should not be considered as anything more than a descriptive example.

HTML defines a special element "BASE" which, when present in the "HEAD" portion of a document, signals that the parser should use the BASE element's "HREF" attribute as the base URI for resolving any relative URI. The "HREF" attribute must be an absolute URI. Note that, in HTML, element and attribute names are case-insensitive. For example:

```
<!doctype html public "-//IETF//DTD HTML//EN">
<HTML><HEAD>
<TITLE>An example HTML document</TITLE>
<BASE href="http://www.ics.uci.edu/Test/a/b/c">
</HEAD><BODY>
... <A href="../x">a hypertext anchor</A> ...
</BODY></HTML>
```

A parser reading the example document should interpret the given relative URI "../x" as representing the absolute URI

```
<http://www.ics.uci.edu/Test/a/x>
```

regardless of the context in which the example document was obtained.

E. Recommendations for Delimiting URI in Context

URI are often transmitted through formats that do not provide a clear context for their interpretation. For example, there are many occasions when URI are included in plain text; examples include text sent in electronic mail, USENET news messages, and, most importantly, printed on paper. In such cases, it is important to be able to delimit the URI from the rest of the text, and in particular from punctuation marks that might be mistaken for part of the URI.

In practice, URI are delimited in a variety of ways, but usually within double-quotes "http://test.com/", angle brackets <http://test.com/>, or just using whitespace

```
                        http://test.com/
```

These wrappers do not form part of the URI.

In the case where a fragment identifier is associated with a URI reference, the fragment would be placed within the brackets as well (separated from the URI with a "#" character).

In some cases, extra whitespace (spaces, linebreaks, tabs, etc.) may need to be added to break long URI across lines. The whitespace should be ignored when extracting the URI.

No whitespace should be introduced after a hyphen ("-") character. Because some typesetters and printers may (erroneously) introduce a hyphen at the end of line when breaking a line, the interpreter of a URI containing a line break immediately after a hyphen should ignore all unescaped whitespace around the line break, and should be aware that the hyphen may or may not actually be part of the URI.

Using <> angle brackets around each URI is especially recommended as a delimiting style for URI that contain whitespace.

The prefix "URL:" (with or without a trailing space) was recommended as a way to used to help distinguish a URL from other bracketed designators, although this is not common in practice.

For robustness, software that accepts user-typed URI should attempt to recognize and strip both delimiters and embedded whitespace.

For example, the text:

Yes, Jim, I found it under "http://www.w3.org/Addressing/", but you can probably pick it up from <ftp://ds.internic.net/rfc/>. Note the warning in <http://www.ics.uci.edu/pub/ietf/uri/historical.html#WARNING>.

contains the URI references

http://www.w3.org/Addressing/
ftp://ds.internic.net/rfc/
http://www.ics.uci.edu/pub/ietf/uri/historical.html#WARNING

F. Abbreviated URLs

The URL syntax was designed for unambiguous reference to network resources and extensibility via the URL scheme. However, as URL identification and usage have become commonplace, traditional media (television, radio, newspapers, billboards, etc.) have increasingly used abbreviated URL references. That is, a reference consisting of only the authority and path portions of the identified resource, such as

www.w3.org/Addressing/

or simply the DNS hostname on its own. Such references are primarily intended for human interpretation rather than machine, with the assumption that context-based heuristics are sufficient to complete the URL (e.g., most hostnames beginning with "www" are likely to have a URL prefix of "http://"). Although there is no standard set of heuristics for disambiguating abbreviated URL references, many client implementations allow them to be entered by the user and heuristically resolved. It should be noted that such heuristics may change over time, particularly when new URL schemes are introduced.

Since an abbreviated URL has the same syntax as a relative URL path, abbreviated URL references cannot be used in contexts where relative URLs are expected. This limits the use of abbreviated URLs to places where there is no defined base URL, such as dialog boxes and off-line advertisements.

G. Summary of Non-editorial Changes

G.1. Additions

Section 4 (URI References) was added to stem the confusion regarding "what is a URI" and how to describe fragment identifiers given that they are not part of the URI, but are part of the URI syntax and parsing concerns. In addition, it provides a reference definition for use by other IETF specifications (HTML, HTTP, etc.) that have previously attempted to redefine the URI syntax in order to account for the presence of fragment identifiers in URI references.

Section 2.4 was rewritten to clarify a number of misinterpretations and to leave room for fully internationalized URI.

Appendix F on abbreviated URLs was added to describe the shortened references often seen on television and magazine advertisements and explain why they are not used in other contexts.

G.2. Modifications from both RFC 1738 and RFC 1808

Changed to URI syntax instead of just URL.

Confusion regarding the terms "character encoding", the URI "character set", and the escaping of characters with %<hex><hex> equivalents has (hopefully) been reduced. Many of the BNF rule names regarding the character sets have been changed to more accurately describe their purpose and to encompass all "characters" rather than just US-ASCII octets. Unless otherwise noted here, these modifications do not affect the URI syntax.

Both RFC 1738 and RFC 1808 refer to the "reserved" set of characters as if URI-interpreting software were limited to a single set of characters with a reserved purpose (i.e., as meaning something other than the data to which the characters correspond), and that this set was fixed by the URI scheme. However, this has not been true in practice; any character that is interpreted differently when it is escaped is, in effect, reserved. Furthermore, the interpreting engine on a HTTP server is often dependent on the resource, not just the URI scheme. The description of reserved characters has been changed accordingly.

The plus "+", dollar "$", and comma "," characters have been added to those in the "reserved" set, since they are treated as reserved within the query component.

The tilde "~" character was added to those in the "unreserved" set, since it is extensively used on the Internet in spite of the difficulty to transcribe it with some keyboards.

The syntax for URI scheme has been changed to require that all schemes begin with an alpha character.

The "user:password" form in the previous BNF was changed to a "userinfo" token, and the possibility that it might be "user:password" made scheme specific. In particular, the use of passwords in the clear is not even suggested by the syntax.

The question-mark "?" character was removed from the set of allowed characters for the userinfo in the authority component, since testing showed that many applications treat it as reserved for separating the query component from the rest of the URI.

The semicolon ";" character was added to those stated as being reserved within the authority component, since several new schemes are using it as a separator within userinfo to indicate the type of user authentication.

RFC 1738 specified that the path was separated from the authority portion of a URI by a slash. RFC 1808 followed suit, but with a fudge of carrying around the separator as a "prefix" in order to describe the parsing algorithm. RFC 1630 never had this problem, since it considered the slash to be part of the path. In writing this specification, it was found to be impossible to accurately describe and retain the difference between the two URI

<foo:/bar> and <foo:bar>

without either considering the slash to be part of the path (as corresponds to actual practice) or creating a separate component just to hold that slash. We chose the former.

G.3. Modifications from RFC 1738

The definition of specific URL schemes and their scheme-specific syntax and semantics has been moved to separate documents.

The URL host was defined as a fully-qualified domain name. However, many URLs are used without fully-qualified domain names (in contexts for which the full qualification is not necessary), without any host (as in some file URLs), or with a host of "localhost".

The URL port is now *digit instead of 1*digit, since systems are expected to handle the case where the ":" separator between host and port is supplied without a port.

The recommendations for delimiting URI in context (Appendix E) have been adjusted to reflect current practice.

G.4. Modifications from RFC 1808

RFC 1808 (Section 4) defined an empty URL reference (a reference containing nothing aside from the fragment identifier) as being a reference to the base URL. Unfortunately, that definition could be interpreted, upon selection of such a reference, as a new retrieval action on that resource. Since the normal intent of such references is for the user agent to change its view of the current document to the beginning of the specified fragment within that document, not to make an additional request of the resource, a description of how to correctly interpret an empty reference has been added in Section 4.

The description of the mythical Base header field has been replaced with a reference to the Content-Location header field defined by MHTML [RFC2110].

RFC 1808 described various schemes as either having or not having the properties of the generic URI syntax. However, the only requirement is that the particular document containing the relative references have a base URI that abides by the generic URI syntax, regardless of the URI scheme, so the associated description has been updated to reflect that.

The BNF term <net_loc> has been replaced with <authority>, since the latter more accurately describes its use and purpose. Likewise, the authority is no longer restricted to the IP server syntax.

Extensive testing of current client applications demonstrated that the majority of deployed systems do not use the ";" character to indicate trailing parameter information, and that the presence of a semicolon in a path segment does not affect the relative parsing of that segment. Therefore, parameters have been removed as a separate component and may now appear in any path segment. Their influence has been removed from the algorithm for resolving a relative URI reference. The resolution examples in Appendix C have been modified to reflect this change.

Implementations are now allowed to work around misformed relative references that are prefixed by the same scheme as the base URI, but only for schemes known to use the <hier_part> syntax.

H. Full Copyright Statement

PORT NUMBERS

(last updated 2001 Aug 27)

The port numbers are divided into three ranges: the Well Known Ports, the Registered Ports, and the Dynamic and/or Private Ports.

The Well Known Ports are those from 0 through 1023.

The Registered Ports are those from 1024 through 49151

The Dynamic and/or Private Ports are those from 49152 through 65535

WELL KNOWN PORT NUMBERS

The Well Known Ports are assigned by the IANA and on most systems can only be used by system (or root) processes or by programs executed by privileged users.

Ports are used in the TCP [RFC793] to name the ends of logical connections which carry long term conversations. For the purpose of providing services to unknown callers, a service contact port is defined. This list specifies the port used by the server process as its contact port. The contact port is sometimes called the "well-known port".

To the extent possible, these same port assignments are used with the UDP [RFC768].

The range for assigned ports managed by the IANA is 0-1023.

Port Assignments:

Keyword	Decimal	Description	References
	0/tcp	Reserved	
	0/udp	Reserved	
#		Jon Postel <postel@isi.edu>	
tcpmux	1/tcp	TCP Port Service Multiplexer	
tcpmux	1/udp	TCP Port Service Multiplexer	
#		Mark Lottor <MKL@nisc.sri.com>	
compressnet	2/tcp	Management Utility	
compressnet	2/udp	Management Utility	
compressnet	3/tcp	Compression Process	
compressnet	3/udp	Compression Process	
#		Bernie Volz <VOLZ@PROCESS.COM>	
#	4/tcp	Unassigned	
#	4/udp	Unassigned	
rje	5/tcp	Remote Job Entry	
rje	5/udp	Remote Job Entry	
#		Jon Postel <postel@isi.edu>	
#	6/tcp	Unassigned	
#	6/udp	Unassigned	
echo	7/tcp	Echo	
echo	7/udp	Echo	
#		Jon Postel <postel@isi.edu>	
#	8/tcp	Unassigned	
#	8/udp	Unassigned	
discard	9/tcp	Discard	
discard	9/udp	Discard	
#		Jon Postel <postel@isi.edu>	
#	10/tcp	Unassigned	
#	10/udp	Unassigned	

systat	11/tcp	Active Users
systat	11/udp	Active Users
#		Jon Postel <postel@isi.edu>
#	12/tcp	Unassigned
#	12/udp	Unassigned
daytime	13/tcp	Daytime (RFC 867)
daytime	13/udp	Daytime (RFC 867)
#		Jon Postel <postel@isi.edu>
#	14/tcp	Unassigned
#	14/udp	Unassigned
#	15/tcp	Unassigned [was netstat]
#	15/udp	Unassigned
#	16/tcp	Unassigned
#	16/udp	Unassigned
qotd	17/tcp	Quote of the Day
qotd	17/udp	Quote of the Day
#		Jon Postel <postel@isi.edu>
msp	18/tcp	Message Send Protocol
msp	18/udp	Message Send Protocol
#		Rina Nethaniel <---none--->
chargen	19/tcp	Character Generator
chargen	19/udp	Character Generator
ftp-data	20/tcp	File Transfer [Default Data]
ftp-data	20/udp	File Transfer [Default Data]
ftp	21/tcp	File Transfer [Control]
ftp	21/udp	File Transfer [Control]
#		Jon Postel <postel@isi.edu>
ssh	22/tcp	SSH Remote Login Protocol
ssh	22/udp	SSH Remote Login Protocol
#		Tatu Ylonen <ylo@cs.hut.fi>
telnet	23/tcp	Telnet
telnet	23/udp	Telnet
#		Jon Postel <postel@isi.edu>
	24/tcp	any private mail system
	24/udp	any private mail system
#		Rick Adams <rick@UUNET.UU.NET>
smtp	25/tcp	Simple Mail Transfer
smtp	25/udp	Simple Mail Transfer
#		Jon Postel <postel@isi.edu>
#	26/tcp	Unassigned
#	26/udp	Unassigned
nsw-fe	27/tcp	NSW User System FE
nsw-fe	27/udp	NSW User System FE
#		Robert Thomas <BThomas@F.BBN.COM>
#	28/tcp	Unassigned
#	28/udp	Unassigned
msg-icp	29/tcp	MSG ICP
msg-icp	29/udp	MSG ICP
#		Robert Thomas <BThomas@F.BBN.COM>
#	30/tcp	Unassigned
#	30/udp	Unassigned
msg-auth	31/tcp	MSG Authentication
msg-auth	31/udp	MSG Authentication
#		Robert Thomas <BThomas@F.BBN.COM>
#	32/tcp	Unassigned
#	32/udp	Unassigned
dsp	33/tcp	Display Support Protocol
dsp	33/udp	Display Support Protocol
#		Ed Cain <cain@edn-unix.dca.mil>
#	34/tcp	Unassigned
#	34/udp	Unassigned
	35/tcp	any private printer server
	35/udp	any private printer server
#		Jon Postel <postel@isi.edu>
#	36/tcp	Unassigned
#	36/udp	Unassigned
time	37/tcp	Time

```
time            37/udp    Time
#                         Jon Postel <postel@isi.edu>
rap             38/tcp    Route Access Protocol
rap             38/udp    Route Access Protocol
#                         Robert Ullmann <ariel@world.std.com>
rlp             39/tcp    Resource Location Protocol
rlp             39/udp    Resource Location Protocol
#                         Mike Accetta <MIKE.ACCETTA@CMU-CS-A.EDU>
#               40/tcp    Unassigned
#               40/udp    Unassigned
graphics        41/tcp    Graphics
graphics        41/udp    Graphics
name            42/tcp    Host Name Server
name            42/udp    Host Name Server
nameserver      42/tcp    Host Name Server
nameserver      42/udp    Host Name Server
nicname         43/tcp    Who Is
nicname         43/udp    Who Is
mpm-flags       44/tcp    MPM FLAGS Protocol
mpm-flags       44/udp    MPM FLAGS Protocol
mpm             45/tcp    Message Processing Module [recv]
mpm             45/udp    Message Processing Module [recv]
mpm-snd         46/tcp    MPM [default send]
mpm-snd         46/udp    MPM [default send]
#                         Jon Postel <postel@isi.edu>
ni-ftp          47/tcp    NI FTP
ni-ftp          47/udp    NI FTP
#                         Steve Kille <S.Kille@isode.com>
auditd          48/tcp    Digital Audit Daemon
auditd          48/udp    Digital Audit Daemon
#                         Larry Scott <scott@zk3.dec.com>
tacacs          49/tcp    Login Host Protocol (TACACS)
tacacs          49/udp    Login Host Protocol (TACACS)
#                         Pieter Ditmars <pditmars@BBN.COM>
re-mail-ck      50/tcp    Remote Mail Checking Protocol
re-mail-ck      50/udp    Remote Mail Checking Protocol
#                         Steve Dorner <s-dorner@UIUC.EDU>
la-maint        51/tcp    IMP Logical Address Maintenance
la-maint        51/udp    IMP Logical Address Maintenance
#                         Andy Malis <malis_a@timeplex.com>
xns-time        52/tcp    XNS Time Protocol
xns-time        52/udp    XNS Time Protocol
#                         Susie Armstrong <Armstrong.wbst128@XEROX>
domain          53/tcp    Domain Name Server
domain          53/udp    Domain Name Server
#                         Paul Mockapetris <PVM@ISI.EDU>
xns-ch          54/tcp    XNS Clearinghouse
xns-ch          54/udp    XNS Clearinghouse
#                         Susie Armstrong <Armstrong.wbst128@XEROX>
isi-gl          55/tcp    ISI Graphics Language
isi-gl          55/udp    ISI Graphics Language
xns-auth        56/tcp    XNS Authentication
xns-auth        56/udp    XNS Authentication
#                         Susie Armstrong <Armstrong.wbst128@XEROX>
                57/tcp    any private terminal access
                57/udp    any private terminal access
#                         Jon Postel <postel@isi.edu>
xns-mail        58/tcp    XNS Mail
xns-mail        58/udp    XNS Mail
#                         Susie Armstrong <Armstrong.wbst128@XEROX>
                59/tcp    any private file service
                59/udp    any private file service
#                         Jon Postel <postel@isi.edu>
                60/tcp    Unassigned
                60/udp    Unassigned
ni-mail         61/tcp    NI MAIL
ni-mail         61/udp    NI MAIL
```



```
#                         Steve Kille <S.Kille@isode.com>
acas            62/tcp    ACA Services
acas            62/udp    ACA Services
#                         E. Wald <ewald@via.enet.dec.com>
whois++         63/tcp    whois++
whois++         63/udp    whois++
#                         Rickard Schoultz <schoultz@sunet.se>
covia           64/tcp    Communications Integrator (CI)
covia           64/udp    Communications Integrator (CI)
#                         Dan Smith <dan.smith@den.galileo.com>
tacacs-ds       65/tcp    TACACS-Database Service
tacacs-ds       65/udp    TACACS-Database Service
#                         Kathy Huber <khuber@bbn.com>
sql*net         66/tcp    Oracle SQL*NET
sql*net         66/udp    Oracle SQL*NET
#                         Jack Haverty <jhaverty@ORACLE.COM>
bootps          67/tcp    Bootstrap Protocol Server
bootps          67/udp    Bootstrap Protocol Server
bootpc          68/tcp    Bootstrap Protocol Client
bootpc          68/udp    Bootstrap Protocol Client
#                         Bill Croft <Croft@SUMEX-AIM.STANFORD.EDU>
tftp            69/tcp    Trivial File Transfer
tftp            69/udp    Trivial File Transfer
#                         David Clark <ddc@LCS.MIT.EDU>
gopher          70/tcp    Gopher
gopher          70/udp    Gopher
#                         Mark McCahill <mpm@boombox.micro.umn.edu>
netrjs-1        71/tcp    Remote Job Service
netrjs-1        71/udp    Remote Job Service
netrjs-2        72/tcp    Remote Job Service
netrjs-2        72/udp    Remote Job Service
netrjs-3        73/tcp    Remote Job Service
netrjs-3        73/udp    Remote Job Service
netrjs-4        74/tcp    Remote Job Service
netrjs-4        74/udp    Remote Job Service
#                         Bob Braden <Braden@ISI.EDU>
                75/tcp    any private dial out service
                75/udp    any private dial out service
#                         Jon Postel <postel@isi.edu>
deos            76/tcp    Distributed External Object Store
deos            76/udp    Distributed External Object Store
#                         Robert Ullmann <ariel@world.std.com>
                77/tcp    any private RJE service
                77/udp    any private RJE service
#                         Jon Postel <postel@isi.edu>
vettcp          78/tcp    vettcp
vettcp          78/udp    vettcp
#                         Christopher Leong <leong@kolmod.mlo.dec.com>
finger          79/tcp    Finger
finger          79/udp    Finger
#                         David Zimmerman <dpz@RUTGERS.EDU>
http            80/tcp    World Wide Web HTTP
http            80/udp    World Wide Web HTTP
www             80/tcp    World Wide Web HTTP
www             80/udp    World Wide Web HTTP
www-http        80/tcp    World Wide Web HTTP
www-http        80/udp    World Wide Web HTTP
#                         Tim Berners-Lee <timbl@W3.org>
hosts2-ns       81/tcp    HOSTS2 Name Server
hosts2-ns       81/udp    HOSTS2 Name Server
#                         Earl Killian <EAK@MORDOR.S1.GOV>
xfer            82/tcp    XFER Utility
xfer            82/udp    XFER Utility
#                         Thomas M. Smith <Thomas.M.Smith@lmco.com>
mit-ml-dev      83/tcp    MIT ML Device
mit-ml-dev      83/udp    MIT ML Device
#                         David Reed <--none--->
```

```
ctf             84/tcp    Common Trace Facility
ctf             84/udp    Common Trace Facility
#                         Hugh Thomas <thomas@oils.enet.dec.com>
mit-ml-dev      85/tcp    MIT ML Device
mit-ml-dev      85/udp    MIT ML Device
#                         David Reed <--none--->
mfcobol         86/tcp    Micro Focus Cobol
mfcobol         86/udp    Micro Focus Cobol
#                         Simon Edwards <--none--->
                87/tcp    any private terminal link
                87/udp    any private terminal link
#                         Jon Postel <postel@isi.edu>
kerberos        88/tcp    Kerberos
kerberos        88/udp    Kerberos
#                         B. Clifford Neuman <bcn@isi.edu>
su-mit-tg       89/tcp    SU/MIT Telnet Gateway
su-mit-tg       89/udp    SU/MIT Telnet Gateway
#                         Mark Crispin <MRC@PANDA.COM>
########## PORT 90 also being used unofficially by Pointcast #########
dnsix           90/tcp    DNSIX Securit Attribute Token Map
dnsix           90/udp    DNSIX Securit Attribute Token Map
#                         Charles Watt <watt@sware.com>
mit-dov         91/tcp    MIT Dover Spooler
mit-dov         91/udp    MIT Dover Spooler
#                         Eliot Moss <EBM@XX.LCS.MIT.EDU>
npp             92/tcp    Network Printing Protocol
npp             92/udp    Network Printing Protocol
#                         Louis Mamakos <louie@sayshell.umd.edu>
dcp             93/tcp    Device Control Protocol
dcp             93/udp    Device Control Protocol
#                         Daniel Tappan <Tappan@BBN.COM>
objcall         94/tcp    Tivoli Object Dispatcher
objcall         94/udp    Tivoli Object Dispatcher
#                         Tom Bereiter <--none--->
supdup          95/tcp    SUPDUP
supdup          95/udp    SUPDUP
#                         Mark Crispin <MRC@PANDA.COM>
dixie           96/tcp    DIXIE Protocol Specification
dixie           96/udp    DIXIE Protocol Specification
#               Tim Howes <Tim.Howes@terminator.cc.umich.edu>
swift-rvf       97/tcp    Swift Remote Virtural File Protocol
swift-rvf       97/udp    Swift Remote Virtural File Protocol
#                         Maurice R. Turcotte
#               <mailrus!uflorida!rm1!dnmrt%rmatl@uunet.UU.NET>
tacnews         98/tcp    TAC News
tacnews         98/udp    TAC News
#                         Jon Postel <postel@isi.edu>
metagram        99/tcp    Metagram Relay
metagram        99/udp    Metagram Relay
#                         Geoff Goodfellow <Geoff@FERNWOOD.MPK.CA.US>
newacct        100/tcp    [unauthorized use]
hostname       101/tcp    NIC Host Name Server
hostname       101/udp    NIC Host Name Server
#                         Jon Postel <postel@isi.edu>
iso-tsap       102/tcp    ISO-TSAP Class 0
iso-tsap       102/udp    ISO-TSAP Class 0
#                         Marshall Rose <mrose@dbc.mtview.ca.us>
gppitnp        103/tcp    Genesis Point-to-Point Trans Net
gppitnp        103/udp    Genesis Point-to-Point Trans Net
acr-nema       104/tcp    ACR-NEMA Digital Imag. & Comm. 300
acr-nema       104/udp    ACR-NEMA Digital Imag. & Comm. 300
#                         Patrick McNamee <--none--->
cso            105/tcp    CCSO name server protocol
cso            105/udp    CCSO name server protocol
#                         Martin Hamilton <martin@mrrl.lut.as.uk>
csnet-ns       105/tcp    Mailbox Name Nameserver
csnet-ns       105/udp    Mailbox Name Nameserver
```



```
#                         Marvin Solomon <solomon@CS.WISC.EDU>
3com-tsmux     106/tcp    3COM-TSMUX
3com-tsmux     106/udp    3COM-TSMUX
#                         Jeremy Siegel <jzs@NSD.3Com.COM>
##########     106        Unauthorized use by insecure poppassd protocol
rtelnet        107/tcp    Remote Telnet Service
rtelnet        107/udp    Remote Telnet Service
#                         Jon Postel <postel@isi.edu>
snagas         108/tcp    SNA Gateway Access Server
snagas         108/udp    SNA Gateway Access Server
#                         Kevin Murphy <murphy@sevens.lkg.dec.com>
pop2           109/tcp    Post Office Protocol - Version 2
pop2           109/udp    Post Office Protocol - Version 2
#                         Joyce K. Reynolds <jkrey@isi.edu>
pop3           110/tcp    Post Office Protocol - Version 3
pop3           110/udp    Post Office Protocol - Version 3
#                         Marshall Rose <mrose@dbc.mtview.ca.us>
sunrpc         111/tcp    SUN Remote Procedure Call
sunrpc         111/udp    SUN Remote Procedure Call
#                         Chuck McManis <cmcmanis@freegate.net>
mcidas         112/tcp    McIDAS Data Transmission Protocol
mcidas         112/udp    McIDAS Data Transmission Protocol
#                         Glenn Davis <support@unidata.ucar.edu>
ident          113/tcp
auth           113/tcp    Authentication Service
auth           113/udp    Authentication Service
#                         Mike St. Johns <stjohns@arpa.mil>
audionews      114/tcp    Audio News Multicast
audionews      114/udp    Audio News Multicast
#                         Martin Forssen <maf@dtek.chalmers.se>
sftp           115/tcp    Simple File Transfer Protocol
sftp           115/udp    Simple File Transfer Protocol
#                         Mark Lottor <MKL@nisc.sri.com>
ansanotify     116/tcp    ANSA REX Notify
ansanotify     116/udp    ANSA REX Notify
#                         Nicola J. Howarth <njh@ansa.co.uk>
uucp-path      117/tcp    UUCP Path Service
uucp-path      117/udp    UUCP Path Service
sqlserv        118/tcp    SQL Services
sqlserv        118/udp    SQL Services
#                         Larry Barnes <barnes@broke.enet.dec.com>
nntp           119/tcp    Network News Transfer Protocol
nntp           119/udp    Network News Transfer Protocol
#                         Phil Lapsley <phil@UCBARPA.BERKELEY.EDU>
cfdptkt        120/tcp    CFDPTKT
cfdptkt        120/udp    CFDPTKT
#                         John Ioannidis <ji@close.cs.columbia.ed>
erpc           121/tcp    Encore Expedited Remote Pro.Call
erpc           121/udp    Encore Expedited Remote Pro.Call
#                         Jack O'Neil <---none--->
smakynet       122/tcp    SMAKYNET
smakynet       122/udp    SMAKYNET
#                         Pierre Arnaud <pierre.arnaud@iname.com>
ntp            123/tcp    Network Time Protocol
ntp            123/udp    Network Time Protocol
#                         Dave Mills <Mills@HUEY.UDEL.EDU>
ansatrader     124/tcp    ANSA REX Trader
ansatrader     124/udp    ANSA REX Trader
#                         Nicola J. Howarth <njh@ansa.co.uk>
locus-map      125/tcp    Locus PC-Interface Net Map Ser
locus-map      125/udp    Locus PC-Interface Net Map Ser
#                         Eric Peterson <lcc.eric@SEAS.UCLA.EDU>
nxedit         126/tcp    NXEdit
nxedit         126/udp    NXEdit
#                          Don Payette <Don.Payette@unisys.com>
##########Port 126 Previously assigned to application below#######
#unitary         126/tcp    Unisys Unitary Login
```

```
#unitary         126/udp    Unisys Unitary Login
#                           <feil@kronos.nisd.cam.unisys.com>
##########Port 126 Previously assigned to application above#######
locus-con        127/tcp    Locus PC-Interface Conn Server
locus-con        127/udp    Locus PC-Interface Conn Server
#                           Eric Peterson <lcc.eric@SEAS.UCLA.EDU>
gss-xlicen       128/tcp    GSS X License Verification
gss-xlicen       128/udp    GSS X License Verification
#                           John Light <johnl@gssc.gss.com>
pwdgen           129/tcp    Password Generator Protocol
pwdgen           129/udp    Password Generator Protocol
#                           Frank J. Wacho <WANCHO@WSMR-SIMTEL20.ARMY.MIL>
cisco-fna        130/tcp    cisco FNATIVE
cisco-fna        130/udp    cisco FNATIVE
cisco-tna        131/tcp    cisco TNATIVE
cisco-tna        131/udp    cisco TNATIVE
cisco-sys        132/tcp    cisco SYSMAINT
cisco-sys        132/udp    cisco SYSMAINT
statsrv          133/tcp    Statistics Service
statsrv          133/udp    Statistics Service
#                           Dave Mills <Mills@HUEY.UDEL.EDU>
ingres-net       134/tcp    INGRES-NET Service
ingres-net       134/udp    INGRES-NET Service
#                           Mike Berrow <---none--->
epmap            135/tcp    DCE endpoint resolution
epmap            135/udp    DCE endpoint resolution
#                           Joe Pato <pato@apollo.hp.com>
profile          136/tcp    PROFILE Naming System
profile          136/udp    PROFILE Naming System
#                           Larry Peterson <llp@ARIZONA.EDU>
netbios-ns       137/tcp    NETBIOS Name Service
netbios-ns       137/udp    NETBIOS Name Service
netbios-dgm      138/tcp    NETBIOS Datagram Service
netbios-dgm      138/udp    NETBIOS Datagram Service
netbios-ssn      139/tcp    NETBIOS Session Service
netbios-ssn      139/udp    NETBIOS Session Service
#                           Jon Postel <postel@isi.edu>
emfis-data       140/tcp    EMFIS Data Service
emfis-data       140/udp    EMFIS Data Service
emfis-cntl       141/tcp    EMFIS Control Service
emfis-cntl       141/udp    EMFIS Control Service
#                           Gerd Beling <GBELING@ISI.EDU>
bl-idm           142/tcp    Britton-Lee IDM
bl-idm           142/udp    Britton-Lee IDM
#                           Susie Snitzer <---none--->
imap             143/tcp    Internet Message Access Protocol
imap             143/udp    Internet Message Access Protocol
#                           Mark Crispin <MRC@CAC.Washington.EDU>
uma              144/tcp    Universal Management Architecture
uma              144/udp    Universal Management Architecture
#                           Jay Whitney <jw@powercenter.com>
uaac             145/tcp    UAAC Protocol
uaac             145/udp    UAAC Protocol
#                           David A. Gomberg <gomberg@GATEWAY.MITRE.ORG>
iso-tp0          146/tcp    ISO-IP0
iso-tp0          146/udp    ISO-IP0
iso-ip           147/tcp    ISO-IP
iso-ip           147/udp    ISO-IP
#                           Marshall Rose <mrose@dbc.mtview.ca.us>
jargon           148/tcp    Jargon
jargon           148/udp    Jargon
#                           Bill Weinman <wew@bearnet.com>
aed-512          149/tcp    AED 512 Emulation Service
aed-512          149/udp    AED 512 Emulation Service
#                           Albert G. Broscius <broscius@DSL.CIS.UPENN.EDU>
sql-net          150/tcp    SQL-NET
sql-net          150/udp    SQL-NET
```



```
#                           Martin Picard <<---none--->
hems             151/tcp    HEMS
hems             151/udp    HEMS
bftp             152/tcp    Background File Transfer Program
bftp             152/udp    Background File Transfer Program
#                           Annette DeSchon <DESCHON@ISI.EDU>
sgmp             153/tcp    SGMP
sgmp             153/udp    SGMP
#                           Marty Schoffstahl <schoff@NISC.NYSER.NET>
netsc-prod       154/tcp    NETSC
netsc-prod       154/udp    NETSC
netsc-dev        155/tcp    NETSC
netsc-dev        155/udp    NETSC
#                           Sergio Heker <heker@JVNCC.CSC.ORG>
sqlsrv           156/tcp    SQL Service
sqlsrv           156/udp    SQL Service
#                           Craig Rogers <Rogers@ISI.EDU>
knet-cmp         157/tcp    KNET/VM Command/Message Protocol
knet-cmp         157/udp    KNET/VM Command/Message Protocol
#                           Gary S. Malkin <GMALKIN@XYLOGICS.COM>
pcmail-srv       158/tcp    PCMail Server
pcmail-srv       158/udp    PCMail Server
#                           Mark L. Lambert <markl@PTT.LCS.MIT.EDU>
nss-routing      159/tcp    NSS-Routing
nss-routing      159/udp    NSS-Routing
#                           Yakov Rekhter <Yakov@IBM.COM>
sgmp-traps       160/tcp    SGMP-TRAPS
sgmp-traps       160/udp    SGMP-TRAPS
#                           Marty Schoffstahl <schoff@NISC.NYSER.NET>
snmp             161/tcp    SNMP
snmp             161/udp    SNMP
snmptrap         162/tcp    SNMPTRAP
snmptrap         162/udp    SNMPTRAP
#                           Marshall Rose <mrose@dbc.mtview.ca.us>
cmip-man         163/tcp    CMIP/TCP Manager
cmip-man         163/udp    CMIP/TCP Manager
cmip-agent       164/tcp    CMIP/TCP Agent
cmip-agent       164/udp    CMIP/TCP Agent
#                           Amatzia Ben-Artzi <---none--->
xns-courier      165/tcp    Xerox
xns-courier      165/udp    Xerox
#                           Susie Armstrong <Armstrong.wbst128@XEROX.COM>
s-net            166/tcp    Sirius Systems
s-net            166/udp    Sirius Systems
#                           Brian Lloyd <brian@lloyd.com>
namp             167/tcp    NAMP
namp             167/udp    NAMP
#                           Marty Schoffstahl <schoff@NISC.NYSER.NET>
rsvd             168/tcp    RSVD
rsvd             168/udp    RSVD
#                           Neil Todd <mcvax!ist.co.uk!neil@UUNET.UU.NET>
send             169/tcp    SEND
send             169/udp    SEND
#                           William D. Wisner <wisner@HAYES.FAI.ALASKA.EDU>
print-srv        170/tcp    Network PostScript
print-srv        170/udp    Network PostScript
#                           Brian Reid <reid@DECWRL.DEC.COM>
multiplex        171/tcp    Network Innovations Multiplex
multiplex        171/udp    Network Innovations Multiplex
cl/1             172/tcp    Network Innovations CL/1
cl/1             172/udp    Network Innovations CL/1
#                           Kevin DeVault <<---none--->
xyplex-mux       173/tcp    Xyplex
xyplex-mux       173/udp    Xyplex
#                           Bob Stewart <STEWART@XYPLEX.COM>
mailq            174/tcp    MAILQ
mailq            174/udp    MAILQ
```

```
#                       Rayan Zachariassen <rayan@AI.TORONTO.EDU>
vmnet           175/tcp VMNET
vmnet           175/udp VMNET
#                       Christopher Tengi <tengi@Princeton.EDU>
genrad-mux      176/tcp GENRAD-MUX
genrad-mux      176/udp GENRAD-MUX
#                       Ron Thornton <thornton@qm7501.genrad.com>
xdmcp           177/tcp X Display Manager Control Protocol
xdmcp           177/udp X Display Manager Control Protocol
#                       Robert W. Scheifler <RWS@XX.LCS.MIT.EDU>
nextstep        178/tcp NextStep Window Server
nextstep        178/udp NextStep Window Server
#                       Leo Hourvitz <leo@NEXT.COM>
bgp             179/tcp Border Gateway Protocol
bgp             179/udp Border Gateway Protocol
#                       Kirk Lougheed <LOUGHEED@MATHOM.CISCO.COM>
ris             180/tcp Intergraph
ris             180/udp Intergraph
#                       Dave Buehmann <ingr!daveb@UUNET.UU.NET>
unify           181/tcp Unify
unify           181/udp Unify
#                       Vinod Singh <--none--->
audit           182/tcp Unisys Audit SITP
audit           182/udp Unisys Audit SITP
#                       Gil Greenbaum <gcole@nisd.cam.unisys.com>
ocbinder        183/tcp OCBinder
ocbinder        183/udp OCBinder
ocserver        184/tcp OCServer
ocserver        184/udp OCServer
#                       Jerrilynn Okamura <--none--->
remote-kis      185/tcp Remote-KIS
remote-kis      185/udp Remote-KIS
kis             186/tcp KIS Protocol
kis             186/udp KIS Protocol
#                       Ralph Droms <rdroms@NRI.RESTON.VA.US>
aci             187/tcp Application Communication Interface
aci             187/udp Application Communication Interface
#                       Rick Carlos <rick.ticipa.csc.ti.com>
mumps           188/tcp Plus Five's MUMPS
mumps           188/udp Plus Five's MUMPS
#                       Hokey Stenn <hokey@PLUS5.COM>
qft             189/tcp Queued File Transport
qft             189/udp Queued File Transport
#                       Wayne Schroeder <schroeder@SDS.SDSC.EDU>
gacp            190/tcp Gateway Access Control Protocol
gacp            190/udp Gateway Access Control Protocol
#                       C. Philip Wood <cpw@LANL.GOV>
prospero        191/tcp Prospero Directory Service
prospero        191/udp Prospero Directory Service
#                       B. Clifford Neuman <bcn@isi.edu>
osu-nms         192/tcp OSU Network Monitoring System
osu-nms         192/udp OSU Network Monitoring System
#                       Doug Karl <KARL-D@OSU-20.IRCC.OHIO-STATE.EDU>
srmp            193/tcp Spider Remote Monitoring Protocol
srmp            193/udp Spider Remote Monitoring Protocol
#                       Ted J. Socolofsky <Teds@SPIDER.CO.UK>
irc             194/tcp Internet Relay Chat Protocol
irc             194/udp Internet Relay Chat Protocol
#                       Jarkko Oikarinen <jto@TOLSUN.OULU.FI>
dn6-nlm-aud     195/tcp DNSIX Network Level Module Audit
dn6-nlm-aud     195/udp DNSIX Network Level Module Audit
dn6-smm-red     196/tcp DNSIX Session Mgt Module Audit Redir
dn6-smm-red     196/udp DNSIX Session Mgt Module Audit Redir
#                       Lawrence Lebahn <DIA3@PAXRV-NES.NAVY.MIL>
dls             197/tcp Directory Location Service
dls             197/udp Directory Location Service
dls-mon         198/tcp Directory Location Service Monitor
```



```
dls-mon         198/udp Directory Location Service Monitor
#                       Scott Bellew <smb@cs.purdue.edu>
smux            199/tcp SMUX
smux            199/udp SMUX
#                       Marshall Rose <mrose@dbc.mtview.ca.us>
src             200/tcp IBM System Resource Controller
src             200/udp IBM System Resource Controller
#                       Gerald McBrearty <---none--->
at-rtmp         201/tcp AppleTalk Routing Maintenance
at-rtmp         201/udp AppleTalk Routing Maintenance
at-nbp          202/tcp AppleTalk Name Binding
at-nbp          202/udp AppleTalk Name Binding
at-3            203/tcp AppleTalk Unused
at-3            203/udp AppleTalk Unused
at-echo         204/tcp AppleTalk Echo
at-echo         204/udp AppleTalk Echo
at-5            205/tcp AppleTalk Unused
at-5            205/udp AppleTalk Unused
at-zis          206/tcp AppleTalk Zone Information
at-zis          206/udp AppleTalk Zone Information
at-7            207/tcp AppleTalk Unused
at-7            207/udp AppleTalk Unused
at-8            208/tcp AppleTalk Unused
at-8            208/udp AppleTalk Unused
#                       Rob Chandhok <chandhok@gnome.cs.cmu.edu>
qmtp            209/tcp The Quick Mail Transfer Protocol
qmtp            209/udp The Quick Mail Transfer Protocol
#                       Dan Bernstein <djb@silverton.berkeley.edu>
z39.50          210/tcp ANSI Z39.50
z39.50          210/udp ANSI Z39.50
#                       Mark Needleman
#                       <mhnur%uccmvsa.bitnet@cornell.cit.cornell.edu>
914c/g          211/tcp Texas Instruments 914C/G Terminal
914c/g          211/udp Texas Instruments 914C/G Terminal
#                       Bill Harrell <---none--->
anet            212/tcp ATEXSSTR
anet            212/udp ATEXSSTR
#                       Jim Taylor <taylor@heart.epps.kodak.com>
ipx             213/tcp IPX
ipx             213/udp IPX
#                       Don Provan <donp@xlnvax.novell.com>
vmpwscs         214/tcp VM PWSCS
vmpwscs         214/udp VM PWSCS
#                       Dan Shia <dset!shia@uunet.UU.NET>
softpc          215/tcp Insignia Solutions
softpc          215/udp Insignia Solutions
#                       Martyn Thomas <---none--->
CAIlic          216/tcp Computer Associates Int'l License Server
CAIlic          216/udp Computer Associates Int'l License Server
#                       Chuck Spitz <spich04@cai.com>
dbase           217/tcp dBASE Unix
dbase           217/udp dBASE Unix
#                       Don Gibson
#             <sequent!aero!twinsun!ashtate.A-T.COM!dong@uunet.UU.NET>
mpp             218/tcp Netix Message Posting Protocol
mpp             218/udp Netix Message Posting Protocol
#                       Shannon Yeh <yeh@netix.com>
uarps           219/tcp Unisys ARPs
uarps           219/udp Unisys ARPs
#                       Ashok Marwaha <---none--->
imap3           220/tcp Interactive Mail Access Protocol v3
imap3           220/udp Interactive Mail Access Protocol v3
#                       James Rice <RICE@SUMEX-AIM.STANFORD.EDU>
fln-spx         221/tcp Berkeley rlogind with SPX auth
fln-spx         221/udp Berkeley rlogind with SPX auth
rsh-spx         222/tcp Berkeley rshd with SPX auth
rsh-spx         222/udp Berkeley rshd with SPX auth
```

```
cdc             223/tcp    Certificate Distribution Center
cdc             223/udp    Certificate Distribution Center
#               Kannan Alagappan <kannan@sejour.enet.dec.com>
########### Possible Conflict of Port 222 with "Masqdialer"##############
### Contact for Masqdialer is Charles Wright <cpwright@villagenet.com>###
masqdialer      224/tcp    masqdialer
masqdialer      224/udp    masqdialer
#                          Charles Wright <cpwright@villagenet.com>
#               225-241    Reserved
#                          Jon Postel <postel@isi.edu>
direct          242/tcp    Direct
direct          242/udp    Direct
#                          Herb Sutter <HerbS@cntc.com>
sur-meas        243/tcp    Survey Measurement
sur-meas        243/udp    Survey Measurement
#                          Dave Clark <ddc@LCS.MIT.EDU>
inbusiness      244/tcp    inbusiness
inbusiness      244/udp    inbusiness
#                          Derrick Hisatake <derrick.i.hisatake@intel.com>
link            245/tcp    LINK
link            245/udp    LINK
dsp3270         246/tcp    Display Systems Protocol
dsp3270         246/udp    Display Systems Protocol
#                          Weldon J. Showalter <Gamma@MINTAKA.DCA.MIL>
subntbcst_tftp  247/tcp    SUBNTBCST_TFTP
subntbcst_tftp  247/udp    SUBNTBCST_TFTP
#                          John Fake <fake@us.ibm.com>
bhfhs           248/tcp    bhfhs
bhfhs           248/udp    bhfhs
#                          John Kelly <johnk@bellhow.com>
#               249-255    Reserved
#                          Jon Postel <postel@isi.edu>
rap             256/tcp    RAP
rap             256/udp    RAP
#                          J.S. Greenfield <greeny@raleigh.ibm.com>
set             257/tcp    Secure Electronic Transaction
set             257/udp    Secure Electronic Transaction
#                          Donald Eastlake <dee3@torque.pothole.com>
yak-chat        258/tcp    Yak Winsock Personal Chat
yak-chat        258/udp    Yak Winsock Personal Chat
#                          Brian Bandy <bbandy@swbell.net>
esro-gen        259/tcp    Efficient Short Remote Operations
esro-gen        259/udp    Efficient Short Remote Operations
#                          Mohsen Banan <mohsen@rostam.neda.com>
openport        260/tcp    Openport
openport        260/udp    Openport
#                          John Marland <jmarland@dean.openport.com>
nsiiops 261/tcp    IIOP Name Service over TLS/SSL
nsiiops         261/udp    IIOP Name Service over TLS/SSL
#                          Jeff Stewart <jstewart@netscape.com>
arcisdms 262/tcp    Arcisdms
arcisdms 262/udp    Arcisdms
#                          Russell Crook (rmc@sni.ca>
hdap            263/tcp    HDAP
hdap            263/udp    HDAP
#                          Troy Gau <troy@zyxel.com>
bgmp            264/tcp    BGMP
bgmp            264/udp    BGMP
#                          Dave Thaler <thalerd@eecs.umich.edu>
x-bone-ctl      265/tcp    X-Bone CTL
x-bone-ctl      265/udp    X-Bone CTL
#                          Joe Touch <touch@isi.edu>
sst             266/tcp    SCSI on ST
sst             266/udp    SCSI on ST
#                          Donald D. Woelz <don@genroco.com>
td-service      267/tcp    Tobit David Service Layer
td-service      267/udp    Tobit David Service Layer
```



```
td-replica      268/tcp    Tobit David Replica
td-replica      268/udp    Tobit David Replica
#                          Franz-Josef Leuders <development@tobit.com>
#               269-279    Unassigned
http-mgmt       280/tcp    http-mgmt
http-mgmt       280/udp    http-mgmt
#                          Adrian Pell
#                          <PELL_ADRIAN/HP-UnitedKingdom_om6@hplb.hpl.hp.com>
personal-link   281/tcp    Personal Link
personal-link   281/udp    Personal Link
#                          Dan Cummings <doc@cnr.com>
cableport-ax    282/tcp    Cable Port A/X
cableport-ax    282/udp    Cable Port A/X
#                          Craig Langfahl <Craig_J_Langfahl@ccm.ch.intel.com>
rescap          283/tcp    rescap
rescap          283/udp    rescap
#                          Paul Hoffman <phoffman@imc.org>
corerjd         284/tcp    corerjd
corerjd         284/udp    corerjd
#                          Chris Thornhill <cjt@corenetworks.com>
#               285        Unassigned
fxp-1           286/tcp    FXP-1
fxp-1           286/udp    FXP-1
#                          James Darnall <jim@cennoid.com>
k-block         287/tcp    K-BLOCK
k-block         287/udp    K-BLOCK
#                          Simon P Jackson <jacko@kring.co.uk>
#               288-307    Unassigned
novastorbakcup  308/tcp    Novastor Backup
novastorbakcup  308/udp    Novastor Backup
#                          Brian Dickman <brian@novastor.com>
entrusttime     309/tcp    EntrustTime
entrusttime     309/udp    EntrustTime
#                          Peter Whittaker <pww@entrust.com>
bhmds           310/tcp    bhmds
bhmds           310/udp    bhmds
#                          John Kelly <johnk@bellhow.com>
asip-webadmin   311/tcp    AppleShare IP WebAdmin
asip-webadmin   311/udp    AppleShare IP WebAdmin
#                          Ann Huang <annhuang@apple.com>
vslmp           312/tcp    VSLMP
vslmp           312/udp    VSLMP
#                          Gerben Wierda <Gerben_Wierda@RnA.nl>
magenta-logic   313/tcp    Magenta Logic
magenta-logic   313/udp    Magenta Logic
#                          Karl Rousseau <kr@netfusion.co.uk>
opalis-robot    314/tcp    Opalis Robot
opalis-robot    314/udp    Opalis Robot
#                          Laurent Domenech, Opalis <ldomenech@opalis.com>
dpsi            315/tcp    DPSI
dpsi            315/udp    DPSI
#                          Tony Scamurra <Tony@DesktopPaging.com>
decauth         316/tcp    decAuth
decauth         316/udp    decAuth
#                          Michael Agishtein <misha@unx.dec.com>
zannet          317/tcp    Zannet
zannet          317/udp    Zannet
#                          Zan Oliphant <zan@accessone.com>
pkix-timestamp  318/tcp    PKIX TimeStamp
pkix-timestamp  318/udp    PKIX TimeStamp
#                          Robert Zuccherato <robert.zuccherato@entrust.com>
ptp-event       319/tcp    PTP Event
ptp-event       319/udp    PTP Event
ptp-general     320/tcp    PTP General
ptp-general     320/udp    PTP General
#                          John Eidson <eidson@hpl.hp.com>
pip             321/tcp    PIP
```

```
pip             321/udp    PIP
#                          Gordon Mohr <gojomo@usa.net>
rtsps           322/tcp    RTSPS
rtsps           322/udp    RTSPS
#                          Anders Klemets <anderskl@microsoft.com>
#               323-332    Unassigned
texar           333/tcp    Texar Security Port
texar           333/udp    Texar Security Port
#                          Eugen Bacic <ebacic@texar.com>
#               334-343    Unassigned
pdap            344/tcp    Prospero Data Access Protocol
pdap            344/udp    Prospero Data Access Protocol
#                          B. Clifford Neuman <bcn@isi.edu>
pawserv         345/tcp    Perf Analysis Workbench
pawserv         345/udp    Perf Analysis Workbench
zserv           346/tcp    Zebra server
zserv           346/udp    Zebra server
fatserv         347/tcp    Fatmen Server
fatserv         347/udp    Fatmen Server
csi-sgwp        348/tcp    Cabletron Management Protocol
csi-sgwp        348/udp    Cabletron Management Protocol
mftp            349/tcp    mftp
mftp            349/udp    mftp
#                          Dave Feinleib <davefe@microsoft.com>
matip-type-a    350/tcp    MATIP Type A
matip-type-a    350/udp    MATIP Type A
matip-type-b    351/tcp    MATIP Type B
matip-type-b    351/udp    MATIP Type B
#                          Alain Robert <arobert@par.sita.int>
# The following entry records an unassigned but widespread use
bhoetty         351/tcp    bhoetty (added 5/21/97)
bhoetty         351/udp    bhoetty
#                          John Kelly <johnk@bellhow.com>
dtag-ste-sb     352/tcp    DTAG (assigned long ago)
dtag-ste-sb     352/udp    DTAG
#                          Ruediger Wald <wald@ez-darmstadt.telekom.de>
# The following entry records an unassigned but widespread use
bhoedap4 352/tcp    bhoedap4 (added 5/21/97)
bhoedap4 352/udp    bhoedap4
#                          John Kelly <johnk@bellhow.com>
ndsauth         353/tcp    NDSAUTH
ndsauth         353/udp    NDSAUTH
#                          Jayakumar Ramalingam <jayakumar@novell.com>
bh611           354/tcp    bh611
bh611           354/udp    bh611
#                          John Kelly <johnk@bellhow.com>
datex-asn       355/tcp    DATEX-ASN
datex-asn       355/udp    DATEX-ASN
#                          Kenneth Vaughn <kvaughn@mail.viggen.com>
cloanto-net-1   356/tcp    Cloanto Net 1
cloanto-net-1   356/udp    Cloanto Net 1
#                          Michael Battilana <mcb@cloanto.com>
bhevent         357/tcp    bhevent
bhevent         357/udp    bhevent
#                          John Kelly <johnk@bellhow.com>
shrinkwrap      358/tcp    Shrinkwrap
shrinkwrap      358/udp    Shrinkwrap
#                          Bill Simpson <wsimpson@greendragon.com>
nsrmp           359/tcp    Network Security Risk Management Protocol
nsrmp           359/udp    Network Security Risk Management Protocol
#                          Eric Jacksch <jacksch@tenebris.ca>
scoi2odialog    360/tcp    scoi2odialog
scoi2odialog    360/udp    scoi2odialog
#                          Keith Petley <keithp@sco.COM>
semantix 361/tcp    Semantix
semantix 361/udp    Semantix
#                          Semantix <xsSupport@semantix.com>
```



```
srssend         362/tcp    SRS Send
srssend         362/udp    SRS Send
#                          Curt Mayer <curt@emergent.com>
rsvp_tunnel     363/tcp    RSVP Tunnel
rsvp_tunnel     363/udp    RSVP Tunnel
#                          Andreas Terzis <terzis@cs.ucla.edu>
aurora-cmgr     364/tcp    Aurora CMGR
aurora-cmgr     364/udp    Aurora CMGR
#                          Philip Budne <budne@auroratech.com>
dtk             365/tcp    DTK
dtk             365/udp    DTK
#                          Fred Cohen <fc@all.net>
odmr            366/tcp    ODMR
odmr            366/udp    ODMR
#                          Randall Gellens <randy@qualcomm.com>
mortgageware    367/tcp    MortgageWare
mortgageware    367/udp    MortgageWare
#                          Ole Hellevik <oleh@interlinq.com>
qbikgdp         368/tcp    QbikGDP
qbikgdp         368/udp    QbikGDP
#                          Adrien de Croy <adrien@qbik.com>
rpc2portmap     369/tcp    rpc2portmap
rpc2portmap     369/udp    rpc2portmap
codaauth2       370/tcp    codaauth2
codaauth2       370/udp    codaauth2
#                          Robert Watson <robert@cyrus.watson.org>
clearcase       371/tcp    Clearcase
clearcase       371/udp    Clearcase
#                          Dave LeBlang <leglang@atria.com>
ulistproc       372/tcp    ListProcessor
ulistproc       372/udp    ListProcessor
#                          Anastasios Kotsikonas <tasos@cs.bu.edu>
legent-1        373/tcp    Legent Corporation
legent-1        373/udp    Legent Corporation
legent-2        374/tcp    Legent Corporation
legent-2        374/udp    Legent Corporation
#                          Keith Boyce <---none--->
hassle          375/tcp    Hassle
hassle          375/udp    Hassle
#                          Reinhard Doelz <doelz@comp.bioz.unibas.ch>
nip             376/tcp    Amiga Envoy Network Inquiry Proto
nip             376/udp    Amiga Envoy Network Inquiry Proto
#                          Heinz Wrobel <hwrobel@gmx.de>
tnETOS          377/tcp    NEC Corporation
tnETOS          377/udp    NEC Corporation
dsETOS          378/tcp    NEC Corporation
dsETOS          378/udp    NEC Corporation
#                          Tomoo Fujita <tf@arc.bs1.fc.nec.co.jp>
is99c           379/tcp    TIA/EIA/IS-99 modem client
is99c           379/udp    TIA/EIA/IS-99 modem client
is99s           380/tcp    TIA/EIA/IS-99 modem server
is99s           380/udp    TIA/EIA/IS-99 modem server
#                          Frank Quick <fquick@qualcomm.com>
hp-collector    381/tcp    hp performance data collector
hp-collector    381/udp    hp performance data collector
hp-managed-node 382/tcp    hp performance data managed node
hp-managed-node 382/udp    hp performance data managed node
hp-alarm-mgr    383/tcp    hp performance data alarm manager
hp-alarm-mgr    383/udp    hp performance data alarm manager
#                          Frank Blakely <frankb@hpptc16.rose.hp.com>
arns            384/tcp    A Remote Network Server System
arns            384/udp    A Remote Network Server System
#                          David Hornsby <djh@munnari.OZ.AU>
ibm-app         385/tcp    IBM Application
ibm-app         385/udp    IBM Application
#                          Lisa Tomita <---none--->
asa             386/tcp    ASA Message Router Object Def.
```

```
asa             386/udp    ASA Message Router Object Def.
#                          Steve Laitinen <laitinen@brutus.aa.ab.com>
aurp            387/tcp    Appletalk Update-Based Routing Pro.
aurp            387/udp    Appletalk Update-Based Routing Pro.
#                          Chris Ranch <cranch@novell.com>
unidata-ldm     388/tcp    Unidata LDM
unidata-ldm     388/udp    Unidata LDM
#                          Glenn Davis <support@unidata.ucar.edu>
#               389/tcp    Lightweight Directory Access Protocol
ldap            389/udp    Lightweight Directory Access Protocol
#                          Tim Howes <Tim.Howes@terminator.cc.umich.edu>
uis             390/tcp    UIS
uis             390/udp    UIS
#                          Ed Barron <---none--->
synotics-relay  391/tcp    SynOptics SNMP Relay Port
synotics-relay  391/udp    SynOptics SNMP Relay Port
synotics-broker 392/tcp    SynOptics Port Broker Port
synotics-broker 392/udp    SynOptics Port Broker Port
#                          Illan Raab <iraab@synoptics.com>
meta5           393/tcp    Meta5
meta5           393/udp    Meta5
#                          Jim Kanzler <jim.kanzler@meta5.com>
embl-ndt        394/tcp    EMBL Nucleic Data Transfer
embl-ndt        394/udp    EMBL Nucleic Data Transfer
#                          Peter Gad <peter@bmc.uu.se>
netcp           395/tcp    NETscout Control Protocol
netcp           395/udp    NETscout Control Protocol
#                          Anil Singhal <---none--->
netware-ip      396/tcp    Novell Netware over IP
netware-ip      396/udp    Novell Netware over IP
mptn            397/tcp    Multi Protocol Trans. Net.
mptn            397/udp    Multi Protocol Trans. Net.
#                          Soumitra Sarkar <sarkar@vnet.ibm.com>
kryptolan       398/tcp    Kryptolan
kryptolan       398/udp    Kryptolan
#                          Peter de Laval <pdl@sectra.se>
iso-tsap-c2     399/tcp    ISO Transport Class 2 Non-Control over TCP
iso-tsap-c2     399/udp    ISO Transport Class 2 Non-Control over TCP
#                          Yanick Pouffary <pouffary@taec.enet.dec.com>
work-sol        400/tcp    Workstation Solutions
work-sol        400/udp    Workstation Solutions
#                          Jim Ward <jimw@worksta.com>
ups             401/tcp    Uninterruptible Power Supply
ups             401/udp    Uninterruptible Power Supply
#                          Charles Bennett <chuck@benatong.com>
genie           402/tcp    Genie Protocol
genie           402/udp    Genie Protocol
#                          Mark Hankin <---none--->
decap           403/tcp    decap
decap           403/udp    decap
nced            404/tcp    nced
nced            404/udp    nced
ncld            405/tcp    ncld
ncld            405/udp    ncld
#                          Richard Jones <---none--->
imsp            406/tcp    Interactive Mail Support Protocol
imsp            406/udp    Interactive Mail Support Protocol
#                          John Myers <jgm+@cmu.edu>
timbuktu        407/tcp    Timbuktu
timbuktu        407/udp    Timbuktu
#                          Marc Epard <marc@netopia.com>
prm-sm          408/tcp    Prospero Resource Manager Sys. Man.
prm-sm          408/udp    Prospero Resource Manager Sys. Man.
prm-nm          409/tcp    Prospero Resource Manager Node Man.
prm-nm          409/udp    Prospero Resource Manager Node Man.
#                          B. Clifford Neuman <bcn@isi.edu>
decladebug      410/tcp    DECLadebug Remote Debug Protocol
```



```
decladebug      410/udp    DECLadebug Remote Debug Protocol
#                          Anthony Berent <anthony.berent@reo.mts.dec.com>
rmt             411/tcp    Remote MT Protocol
rmt             411/udp    Remote MT Protocol
#                          Peter Eriksson <pen@lysator.liu.se>
synoptics-trap  412/tcp    Trap Convention Port
synoptics-trap  412/udp    Trap Convention Port
#                          Illan Raab <iraab@synoptics.com>
smsp            413/tcp    Storage Management Services Protocol
smsp            413/udp    Storage Management Services Protocol
#                          Murthy Srinivas <murthy@novell.com>
infoseek        414/tcp    InfoSeek
infoseek        414/udp    InfoSeek
#                          Steve Kirsch <stk@infoseek.com>
bnet            415/tcp    BNet
bnet            415/udp    BNet
#                          Jim Mertz <JMertz+RV09@rvdc.unisys.com>
silverplatter   416/tcp    Silverplatter
silverplatter   416/udp    Silverplatter
#                          Peter Ciuffetti <petec@silverplatter.com>
onmux           417/tcp    Onmux
onmux           417/udp    Onmux
#                          Stephen Hanna <hanna@world.std.com>
hyper-g         418/tcp    Hyper-G
hyper-g         418/udp    Hyper-G
#                          Frank Kappe <fkappe@iicm.tu-graz.ac.at>
ariel1          419/tcp    Ariel
ariel1          419/udp    Ariel
#                          Lennie Stovel <bl.mds@rlg.org>
smpte           420/tcp    SMPTE
smpte           420/udp    SMPTE
#                          Si Becker <71362.22@CompuServe.COM>
ariel2          421/tcp    Ariel
ariel2          421/udp    Ariel
ariel3          422/tcp    Ariel
ariel3          422/udp    Ariel
#                          Lennie Stovel <bl.mds@rlg.org>
opc-job-start   423/tcp    IBM Operations Planning and Control Start
opc-job-start   423/udp    IBM Operations Planning and Control Start
opc-job-track   424/tcp    IBM Operations Planning and Control Track
opc-job-track   424/udp    IBM Operations Planning and Control Track
#                          Conny Larsson  <cocke@VNET.IBM.COM>
icad-el         425/tcp    ICAD
icad-el         425/udp    ICAD
#                          Larry Stone  <lcs@icad.com>
smartsdp        426/tcp    smartsdp
smartsdp        426/udp    smartsdp
#                          Alexander Dupuy <dupuy@smarts.com>
svrloc          427/tcp    Server Location
svrloc          427/udp    Server Location
#                          <veizades@ftp.com>
ocs_cmu         428/tcp    OCS_CMU
ocs_cmu         428/udp    OCS_CMU
ocs_amu         429/tcp    OCS_AMU
ocs_amu         429/udp    OCS_AMU
#                          Florence Wyman <wyman@peabody.plk.af.mil>
utmpsd          430/tcp    UTMPSD
utmpsd          430/udp    UTMPSD
utmpcd          431/tcp    UTMPCD
utmpcd          431/udp    UTMPCD
iasd            432/tcp    IASD
iasd            432/udp    IASD
#                          Nir Baroz <nbaroz@encore.com>
nnsp            433/tcp    NNSP
nnsp            433/udp    NNSP
#                          Rob Robertson <rob@gangrene.berkeley.edu>
mobileip-agent  434/tcp    MobileIP-Agent
```

```
mobileip-agent  434/udp    MobileIP-Agent
mobilip-mn      435/tcp    MobilIP-MN
mobilip-mn      435/udp    MobilIP-MN
#                          Kannan Alagappan <kannan@sejour.lkg.dec.com>
dna-cml         436/tcp    DNA-CML
dna-cml         436/udp    DNA-CML
#                          Dan Flowers <flowers@smaug.lkg.dec.com>
comscm          437/tcp    comscm
comscm          437/udp    comscm
#                          Jim Teague <teague@zso.dec.com>
dsfgw           438/tcp    dsfgw
dsfgw           438/udp    dsfgw
#                          Andy McKeen <mckeen@osf.org>
dasp            439/tcp    dasp      Thomas Obermair
dasp            439/udp    dasp      tommy@inlab.m.eunet.de
#                          Thomas Obermair <tommy@inlab.m.eunet.de>
sgcp            440/tcp    sgcp
sgcp            440/udp    sgcp
#                          Marshall Rose <mrose@dbc.mtview.ca.us>
decvms-sysmgt   441/tcp    decvms-sysmgt
decvms-sysmgt   441/udp    decvms-sysmgt
#                          Lee Barton <barton@star.enet.dec.com>
cvc_hostd       442/tcp    cvc_hostd
cvc_hostd       442/udp    cvc_hostd
#                          Bill Davidson <billd@equalizer.cray.com>
https           443/tcp    http protocol over TLS/SSL
https           443/udp    http protocol over TLS/SSL
#                          Kipp E.B. Hickman <kipp@mcom.com>
snpp            444/tcp    Simple Network Paging Protocol
snpp            444/udp    Simple Network Paging Protocol
#                          [RFC1568]
microsoft-ds    445/tcp    Microsoft-DS
microsoft-ds    445/udp    Microsoft-DS
#                          Pradeep Bahl <pradeepb@microsoft.com>
ddm-rdb         446/tcp    DDM-RDB
ddm-rdb         446/udp    DDM-RDB
ddm-dfm         447/tcp    DDM-RFM
ddm-dfm         447/udp    DDM-RFM
#                          Jan David Fisher <jdfisher@VNET.IBM.COM>
ddm-ssl         448/tcp    DDM-SSL
ddm-ssl         448/udp    DDM-SSL
#                           Steve Ritland <srr@vnet.ibm.com>
as-servermap    449/tcp    AS Server Mapper
as-servermap    449/udp    AS Server Mapper
#                          Barbara Foss <BGFOSS@rchvmv.vnet.ibm.com>
tserver         450/tcp    TServer
tserver         450/udp    TServer
#                          Harvey S. Schultz <hss@mtgzfs3.mt.att.com>
sfs-smp-net     451/tcp    Cray Network Semaphore server
sfs-smp-net     451/udp    Cray Network Semaphore server
sfs-config      452/tcp    Cray SFS config server
sfs-config      452/udp    Cray SFS config server
#                          Walter Poxon <wdp@ironwood.cray.com>
creativeserver  453/tcp    CreativeServer
creativeserver  453/udp    CreativeServer
contentserver   454/tcp    ContentServer
contentserver   454/udp    ContentServer
creativepartnr  455/tcp    CreativePartnr
creativepartnr  455/udp    CreativePartnr
#                          Jesus Ortiz <jesus_ortiz@emotion.com>
macon-tcp       456/tcp    macon-tcp
macon-udp       456/udp    macon-udp
#                          Yoshinobu Inoue
#                          <shin@hodaka.mfd.cs.fujitsu.co.jp>
scohelp         457/tcp    scohelp
scohelp         457/udp    scohelp
#                          Faith Zack <faithz@sco.com>
```



```
appleqtc        458/tcp    apple quick time
appleqtc        458/udp    apple quick time
#                          Murali Ranganathan
#                          <murali_ranganathan@quickmail.apple.com>
ampr-rcmd       459/tcp    ampr-rcmd
ampr-rcmd       459/udp    ampr-rcmd
#                          Rob Janssen <rob@sys3.pe1chl.ampr.org>
skronk          460/tcp    skronk
skronk          460/udp    skronk
#                          Henry Strickland <strick@yak.net>
datasurfsrv     461/tcp    DataRampSrv
datasurfsrv     461/udp    DataRampSrv
datasurfsrvsec  462/tcp    DataRampSrvSec
datasurfsrvsec  462/udp    DataRampSrvSec
#                          Diane Downie <downie@jibe.MV.COM>
alpes           463/tcp    alpes
alpes           463/udp    alpes
#                          Alain Durand <Alain.Durand@imag.fr>
kpasswd         464/tcp    kpasswd
kpasswd         464/udp    kpasswd
#                          Theodore Ts'o <tytso@MIT.EDU>
urd             465/tcp    URL Rendesvous Directory for SSM
igmpv3lite      465/udp    IGMP over UDP for SSM
#                          Toerless Eckert <eckert@cisco.com>
digital-vrc     466/tcp    digital-vrc
digital-vrc     466/udp    digital-vrc
#                          Peter Higginson <higginson@mail.dec.com>
mylex-mapd      467/tcp    mylex-mapd
mylex-mapd      467/udp    mylex-mapd
#                          Gary Lewis <GaryL@hq.mylex.com>
photuris        468/tcp    proturis
photuris        468/udp    proturis
#                          Bill Simpson <Bill.Simpson@um.cc.umich.edu>
rcp             469/tcp    Radio Control Protocol
rcp             469/udp    Radio Control Protocol
#                          Jim Jennings +1-708-538-7241
scx-proxy       470/tcp    scx-proxy
scx-proxy       470/udp    scx-proxy
#                          Scott Narveson <sjn@cray.com>
mondex          471/tcp    Mondex
mondex          471/udp    Mondex
#                          Bill Reding <redingb@nwdt.natwest.co.uk>
ljk-login       472/tcp    ljk-login
ljk-login       472/udp    ljk-login
#                          LJK Software, Cambridge, Massachusetts
#                          <support@ljk.com>
hybrid-pop      473/tcp    hybrid-pop
hybrid-pop      473/udp    hybrid-pop
#                          Rami Rubin <rami@hybrid.com>
tn-tl-w1        474/tcp    tn-tl-w1
tn-tl-w2        474/udp    tn-tl-w2
#                          Ed Kress <eskress@thinknet.com>
tcpnethaspsrv   475/tcp    tcpnethaspsrv
tcpnethaspsrv   475/udp    tcpnethaspsrv
#                          Charlie Hava <charlie@aladdin.co.il>
tn-tl-fd1       476/tcp    tn-tl-fd1
tn-tl-fd1       476/udp    tn-tl-fd1
#                          Ed Kress <eskress@thinknet.com>
ss7ns           477/tcp    ss7ns
ss7ns           477/udp    ss7ns
#                          Jean-Michel URSCH <ursch@taec.enet.dec.com>
spsc            478/tcp    spsc
spsc            478/udp    spsc
#                          Mike Rieker <mikea@sp32.com>
iafserver       479/tcp    iafserver
iafserver       479/udp    iafserver
iafdbase        480/tcp    iafdbase
```

```
iafdbase        480/udp    iafdbase
#                          ricky@solect.com <Rick Yazwinski>
ph              481/tcp    Ph service
ph              481/udp    Ph service
#                          Roland Hedberg <Roland.Hedberg@umdac.umu.se>
bgs-nsi         482/tcp    bgs-nsi
bgs-nsi         482/udp    bgs-nsi
#                          Jon Saperia <saperia@bgs.com>
ulpnet          483/tcp    ulpnet
ulpnet          483/udp    ulpnet
#                          Kevin Mooney <kevinm@bfs.unibol.com>
integra-sme     484/tcp    Integra Software Management Environment
integra-sme     484/udp    Integra Software Management Environment
#                          Randall Dow <rand@randix.m.isr.de>
powerburst      485/tcp    Air Soft Power Burst
powerburst      485/udp    Air Soft Power Burst
#                          <gary@airsoft.com>
avian           486/tcp    avian
avian           486/udp    avian
#                          Robert Ullmann
#                          <Robert_Ullmann/CAM/Lotus.LOTUS@crd.lotus.com>
saft            487/tcp    saft Simple Asynchronous File Transfer
saft            487/udp    saft Simple Asynchronous File Transfer
#                          Ulli Horlacher <framstag@rus.uni-stuttgart.de>
gss-http        488/tcp    gss-http
gss-http        488/udp    gss-http
#                          Doug Rosenthal <rosenthl@krypton.einet.net>
nest-protocol   489/tcp    nest-protocol
nest-protocol   489/udp    nest-protocol
#                          Gilles Gameiro <ggameiro@birdland.com>
micom-pfs       490/tcp    micom-pfs
micom-pfs       490/udp    micom-pfs
#                          David Misunas <DMisunas@micom.com>
go-login        491/tcp    go-login
go-login        491/udp    go-login
#                          Troy Morrison <troy@graphon.com>
ticf-1          492/tcp    Transport Independent Convergence for FNA
ticf-1          492/udp    Transport Independent Convergence for FNA
ticf-2          493/tcp    Transport Independent Convergence for FNA
ticf-2          493/udp    Transport Independent Convergence for FNA
#                          Mamoru Ito <Ito@pcnet.ks.pfu.co.jp>
pov-ray         494/tcp    POV-Ray
pov-ray         494/udp    POV-Ray
#                          POV-Team Co-ordinator
#                          <iana-port.remove-spamguard@povray.org>
intecourier     495/tcp    intecourier
intecourier     495/udp    intecourier
#                          Steve Favor <sfavor@tigger.intecom.com>
pim-rp-disc     496/tcp    PIM-RP-DISC
pim-rp-disc     496/udp    PIM-RP-DISC
#                          Dino Farinacci <dino@cisco.com>
dantz           497/tcp    dantz
dantz           497/udp    dantz
#                          Richard Zulch <richard_zulch@dantz.com>
siam            498/tcp    siam
siam            498/udp    siam
#                          Philippe Gilbert <pgilbert@cal.fr>
iso-ill         499/tcp    ISO ILL Protocol
iso-ill         499/udp    ISO ILL Protocol
#                          Mark H. Needleman <Mark.Needleman@ucop.edu>
isakmp          500/tcp    isakmp
isakmp          500/udp    isakmp
#                          Mark Schertler <mjs@tycho.ncsc.mil>
stmf            501/tcp    STMF
stmf            501/udp    STMF
#                          Alan Ungar <aungar@farradyne.com>
asa-appl-proto  502/tcp    asa-appl-proto
```



```
asa-appl-proto  502/udp    asa-appl-proto
#                          Dennis Dube <ddube@modicon.com>
intrinsa        503/tcp    Intrinsa
intrinsa        503/udp    Intrinsa
#                          Robert Ford <robert@intrinsa.com>
citadel         504/tcp    citadel
citadel         504/udp    citadel
#                          Art Cancro <ajc@uncnsrd.mt-kisco.ny.us>
mailbox-lm      505/tcp    mailbox-lm
mailbox-lm      505/udp    mailbox-lm
#                          Beverly Moody <Beverly_Moody@stercomm.com>
ohimsrv         506/tcp    ohimsrv
ohimsrv         506/udp    ohimsrv
#                          Scott Powell <spowell@openhorizon.com>
crs             507/tcp    crs
crs             507/udp    crs
#                          Brad Wright <bradwr@microsoft.com>
xvttp           508/tcp    xvttp
xvttp           508/udp    xvttp
#                          Keith J. Alphonso <alphonso@ncs-ssc.com>
snare           509/tcp    snare
snare           509/udp    snare
#                          Dennis Batchelder <dennis@capres.com>
fcp             510/tcp    FirstClass Protocol
fcp             510/udp    FirstClass Protocol
#                          Mike Marshburn <paul@softarc.com>
passgo          511/tcp    PassGo
passgo          511/udp    PassGo
#                          John Rainford <jrainford@passgo.com>
exec            512/tcp    remote process execution;
#                          authentication performed using
#                          passwords and UNIX login names
comsat          512/udp
biff            512/udp    used by mail system to notify users
#                          of new mail received; currently
#                          receives messages only from
#                          processes on the same machine
login           513/tcp    remote login a la telnet;
#                          automatic authentication performed
#                          based on priviledged port numbers
#                          and distributed data bases which
#                          identify "authentication domains"
who             513/udp    maintains data bases showing who's
#                          logged in to machines on a local
#                          net and the load average of the
#                          machine
shell           514/tcp    cmd
#                          like exec, but automatic authentication
#                          is performed as for login server
syslog          514/udp
printer         515/tcp    spooler
printer         515/udp    spooler
videotex        516/tcp    videotex
videotex        516/udp    videotex
#                          Daniel Mavrakis <system@venus.mctel.fr>
talk            517/tcp    like tenex link, but across
#                          machine - unfortunately, doesn't
#                          use link protocol (this is actually
#                          just a rendezvous port from which a
#                          tcp connection is established)
talk            517/udp    like tenex link, but across
#                          machine - unfortunately, doesn't
#                          use link protocol (this is actually
#                          just a rendezvous port from which a
#                          tcp connection is established)
ntalk           518/tcp
ntalk           518/udp
```

```
utime           519/tcp    unixtime
utime           519/udp    unixtime
efs             520/tcp    extended file name server
router          520/udp    local routing process (on site);
#                          uses variant of Xerox NS routing
#                          information protocol - RIP
ripng           521/tcp    ripng
ripng           521/udp    ripng
#                          Robert E. Minnear <minnear@ipsilon.com>
ulp             522/tcp    ULP
ulp             522/udp    ULP
#                          Max Morris <maxm@MICROSOFT.com>
ibm-db2         523/tcp    IBM-DB2
ibm-db2         523/udp    IBM-DB2
#                          Peter Pau <pau@VNET.IBM.COM>
ncp             524/tcp    NCP
ncp             524/udp    NCP
#                          Don Provan <donp@sjf.novell.com>
timed           525/tcp    timeserver
timed           525/udp    timeserver
tempo           526/tcp    newdate
tempo           526/udp    newdate
#                          Unknown
stx             527/tcp    Stock IXChange
stx             527/udp    Stock IXChange
custix          528/tcp    Customer IXChange
custix          528/udp    Customer IXChange
#                          Ferdi Ladeira <ferdi.ladeira@ixchange.com>
irc-serv        529/tcp    IRC-SERV
irc-serv        529/udp    IRC-SERV
#                          Brian Tackett <cym@acrux.net>
courier         530/tcp    rpc
courier         530/udp    rpc
conference      531/tcp    chat
conference      531/udp    chat
netnews         532/tcp    readnews
netnews         532/udp    readnews
netwall         533/tcp    for emergency broadcasts
netwall         533/udp    for emergency broadcasts
mm-admin        534/tcp    MegaMedia Admin
mm-admin        534/udp    MegaMedia Admin
#                          Andreas Heidemann <a.heidemann@ais-gmbh.de>
iiop            535/tcp    iiop
iiop            535/udp    iiop
#                          Jeff M.Michaud <michaud@zk3.dec.com>
opalis-rdv      536/tcp    opalis-rdv
opalis-rdv      536/udp    opalis-rdv
#                          Laurent Domenech <ldomenech@opalis.com>
nmsp            537/tcp    Networked Media Streaming Protocol
nmsp            537/udp    Networked Media Streaming Protocol
#                          Paul Santinelli Jr. <psantinelli@narrative.com>
gdomap          538/tcp    gdomap
gdomap          538/udp    gdomap
#                          Richard Frith-Macdonald <richard@brainstorm.co.uk>
apertus-ldp     539/tcp    Apertus Technologies Load Determination
apertus-ldp     539/udp    Apertus Technologies Load Determination
uucp            540/tcp    uucpd
uucp            540/udp    uucpd
uucp-rlogin     541/tcp    uucp-rlogin
uucp-rlogin     541/udp    uucp-rlogin
#                          Stuart Lynne <sl@wimsey.com>
commerce        542/tcp    commerce
commerce        542/udp    commerce
#                          Randy Epstein <repstein@host.net>
klogin          543/tcp
klogin          543/udp
kshell          544/tcp    krcmd
```



```
kshell          544/udp    krcmd
appleqtcsrvr    545/tcp    appleqtcsrvr
appleqtcsrvr    545/udp    appleqtcsrvr
#                          Murali Ranganathan
#                          <Murali_Ranganathan@quickmail.apple.com>
dhcpv6-client   546/tcp    DHCPv6 Client
dhcpv6-client   546/udp    DHCPv6 Client
dhcpv6-server   547/tcp    DHCPv6 Server
dhcpv6-server   547/udp    DHCPv6 Server
#                          Jim Bound <bound@zk3.dec.com>
afpovertcp      548/tcp    AFP over TCP
afpovertcp      548/udp    AFP over TCP
#                          Leland Wallace <randall@apple.com>
idfp            549/tcp    IDFP
idfp            549/udp    IDFP
#                          Ramana Kovi <ramana@kovi.com>
new-rwho        550/tcp    new-who
new-rwho        550/udp    new-who
cybercash       551/tcp    cybercash
cybercash       551/udp    cybercash
#                          Donald E. Eastlake 3rd <dee@cybercash.com>
deviceshare     552/tcp    deviceshare
deviceshare     552/udp    deviceshare
#                          Brian Schenkenberger <brians@advsyscon.com>
pirp            553/tcp    pirp
pirp            553/udp    pirp
#                          D. J. Bernstein <djb@silverton.berkeley.edu>
rtsp            554/tcp    Real Time Stream Control Protocol
rtsp            554/udp    Real Time Stream Control Protocol
#                          Rob Lanphier <robla@prognet.com>
dsf             555/tcp
dsf             555/udp
remotefs        556/tcp    rfs server
remotefs        556/udp    rfs server
openvms-sysipc  557/tcp    openvms-sysipc
openvms-sysipc  557/udp    openvms-sysipc
#                          Alan Potter <potter@movies.enet.dec.com>
sdnskmp         558/tcp    SDNSKMP
sdnskmp         558/udp    SDNSKMP
teedtap         559/tcp    TEEDTAP
teedtap         559/udp    TEEDTAP
#                          Mort Hoffman <hoffman@mail.ndhm.gtegsc.com>
rmonitor        560/tcp    rmonitord
rmonitor        560/udp    rmonitord
monitor         561/tcp
monitor         561/udp
chshell         562/tcp    chcmd
chshell         562/udp    chcmd
nntps           563/tcp    nntp protocol over TLS/SSL (was snntp)
nntps           563/udp    nntp protocol over TLS/SSL (was snntp)
#                          Kipp E.B. Hickman <kipp@netscape.com>
9pfs            564/tcp    plan 9 file service
9pfs            564/udp    plan 9 file service
whoami          565/tcp    whoami
whoami          565/udp    whoami
streettalk      566/tcp    streettalk
streettalk      566/udp    streettalk
banyan-rpc      567/tcp    banyan-rpc
banyan-rpc      567/udp    banyan-rpc
#                          Tom Lemaire <toml@banyan.com>
ms-shuttle      568/tcp    microsoft shuttle
ms-shuttle      568/udp    microsoft shuttle
#                          Rudolph Balaz <rudolphb@microsoft.com>
ms-rome         569/tcp    microsoft rome
ms-rome         569/udp    microsoft rome
#                          Rudolph Balaz <rudolphb@microsoft.com>
meter           570/tcp    demon
```

```
meter           570/udp    demon
meter           571/tcp    udemon
meter           571/udp    udemon
sonar           572/tcp    sonar
sonar           572/udp    sonar
#                          Keith Moore <moore@cs.utk.edu>
banyan-vip      573/tcp    banyan-vip
banyan-vip      573/udp    banyan-vip
#                          Denis Leclerc <DLeclerc@banyan.com>
ftp-agent       574/tcp    FTP Software Agent System
ftp-agent       574/udp    FTP Software Agent System
#                          Michael S. Greenberg <arnoff@ftp.com>
vemmi           575/tcp    VEMMI
vemmi           575/udp    VEMMI
#                          Daniel Mavrakis <mavrakis@mctel.fr>
ipcd            576/tcp    ipcd
ipcd            576/udp    ipcd
vnas            577/tcp    vnas
vnas            577/udp    vnas
ipdd            578/tcp    ipdd
ipdd            578/udp    ipdd
#                          Jay Farhat <jfarhat@ipass.com>
decbsrv         579/tcp    decbsrv
decbsrv         579/udp    decbsrv
#                          Rudi Martin <movies::martin"@movies.enet.dec.com>
sntp-heartbeat  580/tcp    SNTP HEARTBEAT
sntp-heartbeat  580/udp    SNTP HEARTBEAT
#                          Louis Mamakos <louie@uu.net>
bdp             581/tcp    Bundle Discovery Protocol
bdp             581/udp    Bundle Discovery Protocol
#                          Gary Malkin <gmalkin@xylogics.com>
scc-security    582/tcp    SCC Security
scc-security    582/udp    SCC Security
#                          Prashant Dholakia <prashant@semaphorecom.com>
philips-vc      583/tcp    Philips Video-Conferencing
philips-vc      583/udp    Philips Video-Conferencing
#                          Janna Chang <janna@pmc.philips.com>
keyserver       584/tcp    Key Server
keyserver       584/udp    Key Server
#                          Gary Howland <gary@systemics.com>
imap4-ssl       585/tcp    IMAP4+SSL (use 993 instead)
imap4-ssl       585/udp    IMAP4+SSL (use 993 instead)
#                          Terry Gray <gray@cac.washington.edu>
#              Use of 585 is not recommended, use 993 instead
password-chg    586/tcp    Password Change
password-chg    586/udp    Password Change
submission      587/tcp    Submission
submission      587/udp    Submission
#                          Randy Gellens <randy@qualcomm.com>
cal             588/tcp    CAL
cal             588/udp    CAL
#                          Myron Hattig <Myron_Hattig@ccm.jf.intel.com>
eyelink         589/tcp    EyeLink
eyelink         589/udp    EyeLink
#                          Dave Stampe <dstampe@psych.toronto.edu>
tns-cml         590/tcp    TNS CML
tns-cml         590/udp    TNS CML
#                          Jerome Albin <albin@taec.enet.dec.com>
http-alt 591/tcp    FileMaker, Inc. - HTTP Alternate (see Port 80)
http-alt 591/udp    FileMaker, Inc. - HTTP Alternate (see Port 80)
#                          Clay Maeckel <clay_maeckel@filemaker.com>
eudora-set      592/tcp    Eudora Set
eudora-set      592/udp    Eudora Set
#                          Randall Gellens <randy@qualcomm.com>
http-rpc-epmap  593/tcp    HTTP RPC Ep Map
http-rpc-epmap  593/udp    HTTP RPC Ep Map
#                          Edward Reus <edwardr@microsoft.com>
```



```
tpip            594/tcp    TPIP
tpip            594/udp    TPIP
#                          Brad Spear <spear@platinum.com>
cab-protocol    595/tcp    CAB Protocol
cab-protocol    595/udp    CAB Protocol
#                          Winston Hetherington
smsd            596/tcp    SMSD
smsd            596/udp    SMSD
#                          Wayne Barlow <web@unx.dec.com>
ptcnameservice  597/tcp    PTC Name Service
ptcnameservice  597/udp    PTC Name Service
#                          Yuri Machkasov <yuri@ptc.com>
sco-websrvrmg3  598/tcp    SCO Web Server Manager 3
sco-websrvrmg3  598/udp    SCO Web Server Manager 3
#                          Simon Baldwin <simonb@sco.com>
acp             599/tcp    Aeolon Core Protocol
acp             599/udp    Aeolon Core Protocol
#                          Michael Alyn Miller <malyn@aeolon.com>
ipcserver       600/tcp    Sun IPC server
ipcserver       600/udp    Sun IPC server
#                          Bill Schiefelbein <schief@aspen.cray.com>
#               601-605    Unassigned
urm             606/tcp    Cray Unified Resource Manager
urm             606/udp    Cray Unified Resource Manager
nqs             607/tcp    nqs
nqs             607/udp    nqs
#                          Bill Schiefelbein <schief@aspen.cray.com>
sift-uft        608/tcp    Sender-Initiated/Unsolicited File Transfer
sift-uft        608/udp    Sender-Initiated/Unsolicited File Transfer
#                          Rick Troth <troth@rice.edu>
npmp-trap       609/tcp    npmp-trap
npmp-trap       609/udp    npmp-trap
npmp-local      610/tcp    npmp-local
npmp-local      610/udp    npmp-local
npmp-gui        611/tcp    npmp-gui
npmp-gui        611/udp    npmp-gui
#                          John Barnes <jbarnes@crl.com>
hmmp-ind 612/tcp    HMMP Indication
hmmp-ind 612/udp    HMMP Indication
hmmp-op         613/tcp    HMMP Operation
hmmp-op         613/udp    HMMP Operation
#                          Andrew Sinclair <andrsin@microsoft.com>
sshell          614/tcp    SSLshell
sshell          614/udp    SSLshell
#                          Simon J. Gerraty <sjg@quick.com.au>
sco-inetmgr     615/tcp    Internet Configuration Manager
sco-inetmgr     615/udp    Internet Configuration Manager
sco-sysmgr      616/tcp    SCO System Administration Server
sco-sysmgr      616/udp    SCO System Administration Server
sco-dtmgr       617/tcp    SCO Desktop Administration Server
sco-dtmgr       617/udp    SCO Desktop Administration Server
#                          Christopher Durham <chrisdu@sco.com>
dei-icda 618/tcp    DEI-ICDA
dei-icda 618/udp    DEI-ICDA
#                          David Turner <digital@Quetico.tbaytel.net>
digital-evm     619/tcp    Digital EVM
digital-evm     619/udp    Digital EVM
#                          Jem Treadwell <jem@unx.dec.com>
sco-websrvrmgr  620/tcp    SCO WebServer Manager
sco-websrvrmgr  620/udp    SCO WebServer Manager
#                          Christopher Durham <chrisdu@sco.com>
escp-ip         621/tcp    ESCP
escp-ip         621/udp    ESCP
#                          Lai Zit Seng <lzs@pobox.com>
collaborator    622/tcp    Collaborator
collaborator    622/udp    Collaborator
#                          Johnson Davis <johnsond@opteamasoft.com>
```

```
aux_bus_shunt   623/tcp    Aux Bus Shunt
aux_bus_shunt   623/udp    Aux Bus Shunt
#                          Steve Williams <Steven_D_Williams@ccm.jf.intel.com>
cryptoadmin     624/tcp    Crypto Admin
cryptoadmin     624/udp    Crypto Admin
#                          Tony Walker <tony@cryptocard.com>
dec_dlm         625/tcp    DEC DLM
dec_dlm         625/udp    DEC DLM
#                          Rudi Martin <Rudi.Martin@edo.mts.dec.com>
asia            626/tcp    ASIA
asia            626/udp    ASIA
#                          Michael Dasenbrock <dasenbro@apple.com>
passgo-tivoli   627/tcp    PassGo Tivoli
passgo-tivoli   627/udp    PassGo Tivoli
#                          Chris Hall <chall@passgo.com>
qmqp            628/tcp    QMQP
qmqp            628/udp    QMQP
#                          Dan Bernstein <djb@cr.yp.to>
3com-amp3       629/tcp    3Com AMP3
3com-amp3       629/udp    3Com AMP3
#                          Prakash Banthia <prakash_banthia@3com.com>
rda             630/tcp    RDA
rda             630/udp    RDA
#                          John Hadjioannou <john@minster.co.uk>
ipp             631/tcp    IPP (Internet Printing Protocol)
ipp             631/udp    IPP (Internet Printing Protocol)
#                          Carl-Uno Manros <manros@cp10.es.xerox.com>
bmpp            632/tcp    bmpp
bmpp            632/udp    bmpp
#                          Troy Rollo <troy@kroll.corvu.com.au>
servstat 633/tcp    Service Status update (Sterling Software)
servstat 633/udp    Service Status update (Sterling Software)
#                          Greg Rose <Greg_Rose@sydney.sterling.com>
ginad           634/tcp    ginad
ginad           634/udp    ginad
#                          Mark Crother <mark@eis.calstate.edu>
rlzdbase        635/tcp    RLZ DBase
rlzdbase        635/udp    RLZ DBase
#                          Michael Ginn <ginn@tyxar.com>
ldaps           636/tcp    ldap protocol over TLS/SSL (was sldap)
ldaps           636/udp    ldap protocol over TLS/SSL (was sldap)
#                          Pat Richard <patr@xcert.com>
lanserver       637/tcp    lanserver
lanserver       637/udp    lanserver
#                          Chris Larsson <clarsson@VNET.IBM.COM>
mcns-sec 638/tcp    mcns-sec
mcns-sec 638/udp    mcns-sec
#                          Kaz Ozawa <k.ozawa@cablelabs.com>
msdp            639/tcp    MSDP
msdp            639/udp    MSDP
#                          Dino Farinacci <dino@cisco.com>
entrust-sps     640/tcp    entrust-sps
entrust-sps     640/udp    entrust-sps
#                          Marek Buchler <Marek.Buchler@entrust.com>
repcmd          641/tcp    repcmd
repcmd          641/udp    repcmd
#                          Scott Dale <scott@Replicase.com>
esro-emsdp      642/tcp    ESRO-EMSDP V1.3
esro-emsdp      642/udp    ESRO-EMSDP V1.3
#                          Mohsen Banan <mohsen@neda.com>
sanity          643/tcp    SANity
sanity          643/udp    SANity
#                          Peter Viscarola <PeterGV@osr.com>
dwr             644/tcp    dwr
dwr             644/udp    dwr
#                          Bill Fenner <fenner@parc.xerox.com>
pssc            645/tcp    PSSC
```



```
pssc            645/udp    PSSC
#                          Egon Meier-Engelen <egon.meier-engelen@dlr.de>
ldp             646/tcp    LDP
ldp             646/udp    LDP
#                          Bob Thomas <rhthomas@cisco.com>
dhcp-failover   647/tcp    DHCP Failover
dhcp-failover   647/udp    DHCP Failover
#                          Bernard Volz <volz@ipworks.com>
rrp             648/tcp    Registry Registrar Protocol (RRP)
rrp             648/udp    Registry Registrar Protocol (RRP)
#                          Scott Hollenbeck <shollenb@netsol.com>
aminet          649/tcp    Aminet
aminet          649/udp    Aminet
#                          Martin Toeller <mtoeller@adaptivemedia.com>
obex            650/tcp    OBEX
obex            650/udp    OBEX
#                          Jeff Garbers <FJG030@email.mot.com>
ieee-mms 651/tcp    IEEE MMS
ieee-mms 651/udp    IEEE MMS
#                          Curtis Anderson <canderson@turbolinux.com>
hello-port      652/tcp    HELLO_PORT
hello-port      652/udp    HELLO_PORT
#                          Patrick Cipiere <Patrick.Cipiere@UDcast.com>
repscmd         653/tcp    RepCmd
repscmd         653/udp    RepCmd
#                          Scott Dale <scott@tioga.com>
aodv            654/tcp    AODV
aodv            654/udp    AODV
#                          Charles Perkins <cperkins@eng.sun.com>
tinc            655/tcp    TINC
tinc            655/udp    TINC
#                          Ivo Timmermans <itimmermans@bigfoot.com>
spmp            656/tcp    SPMP
spmp            656/udp    SPMP
#                          Jakob Kaivo <jkaivo@nodomainname.net>
rmc             657/tcp    RMC
rmc             657/udp    RMC
#                          Michael Schmidt <mmaass@us.ibm.com>
tenfold         658/tcp    TenFold
tenfold         658/udp    TenFold
#                          Louis Olszyk <lolszyk@10fold.com>
#               659        De-Registered (2001 June 06)
mac-srvr-admin  660/tcp    MacOS Server Admin
mac-srvr-admin  660/udp    MacOS Server Admin
#                          Forest Hill <forest@apple.com>
hap             661/tcp    HAP
hap             661/udp    HAP
#                          Igor Plotnikov <igor@uroam.com>
pftp            662/tcp    PFTP
pftp            662/udp    PFTP
#                          Ben Schluricke <pftp@star.trek.org>
purenoise       663/tcp    PureNoise
purenoise       663/udp    PureNoise
#                          Sam Osa <pristine@mailcity.com>
secure-aux-bus  664/tcp    Secure Aux Bus
secure-aux-bus  664/udp    Secure Aux Bus
#                          Steven Williams <steven.d.williams@intel.com>
sun-dr          665/tcp    Sun DR
sun-dr          665/udp    Sun DR
#                          Harinder Bhasin <Harinder.Bhasin@Sun.COM>
mdqs            666/tcp
mdqs            666/udp
doom            666/tcp    doom Id Software
doom            666/udp    doom Id Software
#                          <ddt@idcube.idsoftware.com>
disclose        667/tcp    campaign contribution disclosures - SDR Technologies
disclose        667/udp    campaign contribution disclosures - SDR Technologies
```

```
#                       Jim Dixon  <jim@lambda.com>
mecomm          668/tcp    MeComm
mecomm          668/udp    MeComm
meregister      669/tcp    MeRegister
meregister      669/udp    MeRegister
#                       Armin Sawusch <armin@esd1.esd.de>
vacdsm-sws      670/tcp    VACDSM-SWS
vacdsm-sws      670/udp    VACDSM-SWS
vacdsm-app      671/tcp    VACDSM-APP
vacdsm-app      671/udp    VACDSM-APP
vpps-qua        672/tcp    VPPS-QUA
vpps-qua        672/udp    VPPS-QUA
cimplex         673/tcp    CIMPLEX
cimplex         673/udp    CIMPLEX
#                       Ulysses G. Smith Jr. <ugsmith@cesi.com>
acap            674/tcp    ACAP
acap            674/udp    ACAP
#                       Chris Newman <Chris.Newman@innosoft.com>
dctp            675/tcp    DCTP
dctp            675/udp    DCTP
#                       Andre Kramer <Andre.Kramer@ansa.co.uk>
vpps-via 676/tcp    VPPS Via
vpps-via 676/udp    VPPS Via
#                       Ulysses G. Smith Jr. <ugsmith@cesi.com>
vpp             677/tcp    Virtual Presence Protocol
vpp             677/udp    Virtual Presence Protocol
#                       Klaus Wolf <wolf@cobrow.com>
ggf-ncp         678/tcp    GNU Generation Foundation NCP
ggf-ncp         678/udp    GNU Generation Foundation NCP
#                       Noah Paul <noahp@altavista.net>
mrm             679/tcp    MRM
mrm             679/udp    MRM
#                       Liming Wei <lwei@cisco.com>
entrust-aaas    680/tcp    entrust-aaas
entrust-aaas    680/udp    entrust-aaas
entrust-aams    681/tcp    entrust-aams
entrust-aams    681/udp    entrust-aams
#                       Adrian Mancini <adrian.mancini@entrust.com>
xfr             682/tcp    XFR
xfr             682/udp    XFR
#                       Noah Paul <noahp@ultranet.com>
corba-iiop      683/tcp    CORBA IIOP
corba-iiop      683/udp    CORBA IIOP
corba-iiop-ssl  684/tcp    CORBA IIOP SSL
corba-iiop-ssl  684/udp    CORBA IIOP SSL
#                       Henry Lowe <lowe@omg.org>
mdc-portmapper  685/tcp    MDC Port Mapper
mdc-portmapper  685/udp    MDC Port Mapper
#                       Noah Paul <noahp@altavista.net>
hcp-wismar      686/tcp    Hardware Control Protocol Wismar
hcp-wismar      686/udp    Hardware Control Protocol Wismar
#                       David Merchant <d.f.merchant@livjm.ac.uk>
asipregistry    687/tcp    asipregistry
asipregistry    687/udp    asipregistry
#                       Erik Sea <sea@apple.com>
realm-rusd      688/tcp    REALM-RUSD
realm-rusd      688/udp    REALM-RUSD
#                       Jerry Knight <jknight@realminfo.com>
nmap            689/tcp    NMAP
nmap            689/udp    NMAP
#                       Peter Dennis Bartok <peter@novonyx.com>
vatp            690/tcp    VATP
vatp            690/udp    VATP
#                       Atica Software <comercial@aticasoft.es>
msexch-routing  691/tcp    MS Exchange Routing
msexch-routing  691/udp    MS Exchange Routing
#                       David Lemson <dlemson@microsoft.com>
```



```
hyperwave-isp   692/tcp    Hyperwave-ISP
hyperwave-isp   692/udp    Hyperwave-ISP
#                       Gerald Mesaric <gmesaric@hyperwave.com>
connendp 693/tcp    connendp
connendp 693/udp    connendp
#                       Ronny Bremer <rbremer@future-gate.com>
ha-cluster      694/tcp    ha-cluster
ha-cluster      694/udp    ha-cluster
#                       Alan Robertson <alanr@unix.sh>
ieee-mms-ssl    695/tcp    IEEE-MMS-SSL
ieee-mms-ssl    695/udp    IEEE-MMS-SSL
#                       Curtis Anderson <ecanderson@turbolinux.com>
rushd           696/tcp    RUSHD
rushd           696/udp    RUSHD
#                       Greg Ercolano <erco@netcom.com>
uuidgen         697/tcp    UUIDGEN
uuidgen         697/udp    UUIDGEN
#                       James Falkner <jhf@eng.sun.com>
olsr            698/tcp    OLSR
olsr            698/udp    OLSR
#                       Thomas Clausen <thomas.clausen@inria.fr>
accessnetwork   699/tcp    Access Network
accessnetwork   699/udp    Access Network
#                       Yingchun Xu <Yingchun_Xu@3com.com>
#               700-703    Unassigned
elcsd           704/tcp    errlog copy/server daemon
elcsd           704/udp    errlog copy/server daemon
agentx          705/tcp    AgentX
agentx          705/udp    AgentX
#                       Bob Natale <natale@acec.com>
silc            706/tcp    SILC
silc            706/udp    SILC
#                       Pekka Riikonen <priikone@poseidon.pspt.fi>
borland-dsj     707/tcp    Borland DSJ
borland-dsj     707/udp    Borland DSJ
#                       Gerg Cole <gcole@corp.borland.com>
#               708        Unassigned
entrust-kmsh    709/tcp    Entrust Key Management Service Handler
entrust-kmsh    709/udp    Entrust Key Management Service Handler
entrust-ash     710/tcp    Entrust Administration Service Handler
entrust-ash     710/udp    Entrust Administration Service Handler
#                       Peter Whittaker <pww@entrust.com>
cisco-tdp       711/tcp    Cisco TDP
cisco-tdp       711/udp    Cisco TDP
#                       Bruce Davie <bsd@cisco.com>
#               712-728    Unassigned
netviewdm1      729/tcp    IBM NetView DM/6000 Server/Client
netviewdm1      729/udp    IBM NetView DM/6000 Server/Client
netviewdm2      730/tcp    IBM NetView DM/6000 send/tcp
netviewdm2      730/udp    IBM NetView DM/6000 send/tcp
netviewdm3      731/tcp    IBM NetView DM/6000 receive/tcp
netviewdm3      731/udp    IBM NetView DM/6000 receive/tcp
#                       Philippe Binet  (phbinet@vnet.IBM.COM)
#               732-740    Unassigned
netgw           741/tcp    netGW
netgw           741/udp    netGW
#                       Oliver Korfmacher (okorf@netcs.com)
netrcs          742/tcp    Network based Rev. Cont. Sys.
netrcs          742/udp    Network based Rev. Cont. Sys.
#                       Gordon C. Galligher <gorpong@ping.chi.il.us>
#               743        Unassigned
flexlm          744/tcp    Flexible License Manager
flexlm          744/udp    Flexible License Manager
#                       Matt Christiano
#                       <globes@matt@oliveb.atc.olivetti.com>
#               745-746    Unassigned
fujitsu-dev     747/tcp    Fujitsu Device Control
```

```
fujitsu-dev     747/udp    Fujitsu Device Control
ris-cm          748/tcp    Russell Info Sci Calendar Manager
ris-cm          748/udp    Russell Info Sci Calendar Manager
kerberos-adm    749/tcp    kerberos administration
kerberos-adm    749/udp    kerberos administration
rfile           750/tcp
loadav          750/udp
kerberos-iv     750/udp    kerberos version iv
#                          Martin Hamilton <martin@mrrl.lut.as.uk>
pump            751/tcp
pump            751/udp
qrh             752/tcp
qrh             752/udp
rrh             753/tcp
rrh             753/udp
tell            754/tcp    send
tell            754/udp    send
#                          Josyula R. Rao <jrrao@watson.ibm.com>
#               755-756    Unassigned
nlogin          758/tcp
nlogin          758/udp
con             759/tcp
con             759/udp
ns              760/tcp
ns              760/udp
rxe             761/tcp
rxe             761/udp
quotad          762/tcp
quotad          762/udp
cycleserv       763/tcp
cycleserv       763/udp
omserv          764/tcp
omserv          764/udp
webster         765/tcp
webster         765/udp
#                          Josyula R. Rao <jrrao@watson.ibm.com>
#               766        Unassigned
phonebook       767/tcp    phone
phonebook       767/udp    phone
#                          Josyula R. Rao <jrrao@watson.ibm.com>
#               768        Unassigned
vid             769/tcp
vid             769/udp
cadlock         770/tcp
cadlock         770/udp
rtip            771/tcp
rtip            771/udp
cycleserv2      772/tcp
cycleserv2      772/udp
submit          773/tcp
notify          773/udp
rpasswd         774/tcp
acmaint_dbd     774/udp
entomb          775/tcp
acmaint_transd  775/udp
wpages          776/tcp
wpages          776/udp
#                          Josyula R. Rao <jrrao@watson.ibm.com>
multiling-http  777/tcp    Multiling HTTP
multiling-http  777/udp    Multiling HTTP
#                          Alejandro Bonet <babel@ctv.es>
#               778-779    Unassigned
wpgs            780/tcp
wpgs            780/udp
#                          Josyula R. Rao <jrrao@watson.ibm.com>
#               781-785    Unassigned
concert         786/tcp    Concert
```



```
concert         786/udp    Concert
#                          Josyula R. Rao <jrrao@watson.ibm.com>
qsc             787/tcp    QSC
qsc             787/udp    QSC
#                          James Furness <furn@bluenews.com>
#               788-799    Unassigned
mdbs_daemon     800/tcp
mdbs_daemon     800/udp
device          801/tcp
device          801/udp
#               802-809    Unassigned
fcp-udp         810/tcp    FCP
fcp-udp         810/udp    FCP Datagram
#                          Paul Whittemore <paul@softarc.com>
#               811-827    Unassigned
itm-mcell-s     828/tcp    itm-mcell-s
itm-mcell-s     828/udp    itm-mcell-s
#                          Miles O'Neal <meo@us.itmasters.com>
pkix-3-ca-ra    829/tcp    PKIX-3 CA/RA
pkix-3-ca-ra    829/udp    PKIX-3 CA/RA
#                          Carlisle Adams <Cadams@entrust.com>
#               830-846    Unassigned
dhcp-failover2  847/tcp    dhcp-failover 2
dhcp-failover2  847/udp    dhcp-failover 2
#                          Bernard Volz <volz@ipworks.com>
#               848-872    Unassigned
rsync           873/tcp    rsync
rsync           873/udp    rsync
#                          Andrew Tridgell <tridge@samba.anu.edu.au>
#               874-885    Unassigned
iclcnet-locate  886/tcp    ICL coNETion locate server
iclcnet-locate  886/udp    ICL coNETion locate server
#                          Bob Lyon <bl@oasis.icl.co.uk>
iclcnet_svinfo  887/tcp    ICL coNETion server info
iclcnet_svinfo  887/udp    ICL coNETion server info
#                          Bob Lyon <bl@oasis.icl.co.uk>
accessbuilder   888/tcp    AccessBuilder
accessbuilder   888/udp    AccessBuilder
#                          Steve Sweeney <Steven_Sweeney@3mail.3com.com>
# The following entry records an unassigned but widespread use
cddbp           888/tcp    CD Database Protocol
#                          Steve Scherf <steve@moonsoft.com>
#
#               889-899    Unassigned
omginitialrefs  900/tcp    OMG Initial Refs
omginitialrefs  900/udp    OMG Initial Refs
#                          Christian Callsen <Christian.Callsen@eng.sun.com>
smpnameres      901/tcp    SMPNAMERES
smpnameres      901/udp    SMPNAMERES
#                          Leif Ekblad <leif@rdos.net>
ideafarm-chat   902/tcp    IDEAFARM-CHAT
ideafarm-chat   902/udp    IDEAFARM-CHAT
ideafarm-catch  903/tcp    IDEAFARM-CATCH
ideafarm-catch  903/udp    IDEAFARM-CATCH
#                          Wo'o Ideafarm <wo@ideafarm.com>
#               904-910    Unassigned
xact-backup     911/tcp    xact-backup
xact-backup     911/udp    xact-backup
#                          Bill Carroll <billc@xactlabs.com>
#               912-988    Unassigned
ftps-data       989/tcp    ftp protocol, data, over TLS/SSL
ftps-data       989/udp    ftp protocol, data, over TLS/SSL
ftps            990/tcp    ftp protocol, control, over TLS/SSL
ftps            990/udp    ftp protocol, control, over TLS/SSL
#                          Christopher Allen <ChristopherA@consensus.com>
nas             991/tcp    Netnews Administration System
nas             991/udp    Netnews Administration System
```

```
#                          Vera Heinau <heinau@fu-berlin.de>
#                          Heiko Schlichting <heiko@fu-berlin.de>
telnets         992/tcp    telnet protocol over TLS/SSL
telnets         992/udp    telnet protocol over TLS/SSL
imaps           993/tcp    imap4 protocol over TLS/SSL
imaps           993/udp    imap4 protocol over TLS/SSL
ircs            994/tcp    irc protocol over TLS/SSL
ircs            994/udp    irc protocol over TLS/SSL
#                          Christopher Allen <ChristopherA@consensus.com>
pop3s           995/tcp    pop3 protocol over TLS/SSL (was spop3)
pop3s           995/udp    pop3 protocol over TLS/SSL (was spop3)
#                          Gordon Mangione <gordm@microsoft.com>
vsinet          996/tcp    vsinet
vsinet          996/udp    vsinet
#                          Rob Juergens <robj@vsi.com>
maitrd          997/tcp
maitrd          997/udp
busboy          998/tcp
puparp          998/udp
garcon          999/tcp
applix          999/udp        Applix ac
puprouter       999/tcp
puprouter       999/udp
cadlock2 1000/tcp
cadlock2 1000/udp
#               1001-1009      Unassigned
#               1008/udp   Possibly used by Sun Solaris????
surf            1010/tcp       surf
surf            1010/udp       surf
#                              Joseph Geer <jgeer@peapod.com>
#               1011-1022      Reserved
                1023/tcp       Reserved
                1023/udp       Reserved
#                              IANA <iana@iana.org>



REGISTERED PORT NUMBERS

The Registered Ports are listed by the IANA and on most systems can be
used by ordinary user processes or programs executed by ordinary
users.

Ports are used in the TCP [RFC793] to name the ends of logical
connections which carry long term conversations.  For the purpose of
providing services to unknown callers, a service contact port is
defined.  This list specifies the port used by the server process as
its contact port.

The IANA registers uses of these ports as a convenience to the
community.

To the extent possible, these same port assignments are used with the
UDP [RFC768].

The Registered Ports are in the range 1024-49151.
```

The rest of this document is omitted. The full text can be found at http://www.iana.org/assignments/port-numbers.

MEDIA TYPES

(last updated 2001 August 23)

[RFC2045,RFC2046] specifies that Content Types, Content Subtypes, Character Sets, Access Types, and conversion values for MIME mail will be assigned and listed by the IANA.

Content Types and Subtypes

Type	Subtype	Description	Reference
text	plain		[RFC2646,RFC2046]
	richtext		[RFC2045,RFC2046]
	enriched		[RFC1896]
	tab-separated-values		[Paul Lindner]
	html		[RFC2854]
	sgml		[RFC1874]
	vnd.latex-z		[Lubos]
	vnd.fmi.flexstor		[Hurtta]
	uri-list		[RFC2483]
	vnd.abc		[Allen]
	rfc822-headers		[RFC1892]
	vnd.in3d.3dml		[Powers]
	prs.lines.tag		[Lines]
	vnd.in3d.spot		[Powers]
	css		[RFC2318]
	xml		[RFC3023]
	xml-external-parsed-entity		[RFC3023]
	rtf		[Lindner]
	directory		[RFC2425]
	calendar		[RFC2445]
	vnd.wap.wml		[Stark]
	vnd.wap.wmlscript		[Stark]
	vnd.motorola.reflex		[Patton]
	vnd.fly		[Gurney]
	vnd.wap.sl		[WAP-Forum]
	vnd.wap.si		[WAP-Forum]
	t140		[RFC2793]
	vnd.ms-mediapackage		[Nelson]
	vnd.IPTC.NewsML		[IPTC]
	vnd.IPTC.NITF		[IPTC]
	vnd.curl		[Hodge]
	vnd.DMClientScript		[Bradley]
	parityfec		[RFC3009]
multipart	mixed		[RFC2045,RFC2046]
	alternative		[RFC2045,RFC2046]
	digest		[RFC2045,RFC2046]
	parallel		[RFC2045,RFC2046]
	appledouble		[MacMime,Patrik Faltstrom]
	header-set		[Dave Crocker]
	form-data		[RFC2388]
	related		[RFC2387]
	report		[RFC1892]
	voice-message		[RFC2421,RFC2423]
	signed		[RFC1847]
	encrypted		[RFC1847]
	byteranges		[RFC2068]
message	rfc822		[RFC2045,RFC2046]
	partial		[RFC2045,RFC2046]
	external-body		[RFC2045,RFC2046]
	news		[RFC 1036, Henry Spencer]
	http		[RFC2616]
	delivery-status		[RFC1894]
	disposition-notification		[RFC2298]
	s-http		[RFC2660]
application	octet-stream		[RFC2045,RFC2046]
	postscript		[RFC2045,RFC2046]
	oda		[RFC2045,RFC2046]
	atomicmail		[atomicmail,Borenstein]
	andrew-inset		[andrew-inset,Borenstein]
	slate		[slate,terry crowley]
	wita		[Wang Info Transfer,Larry Campbell]
	dec-dx		[Digital Doc Trans, Larry Campbell]
	dca-rft		[IBM Doc Content Arch, Larry Campbell]
	activemessage		[Ehud Shapiro]
	rtf		[Paul Lindner]
	applefile		[MacMime,Patrik Faltstrom]
	mac-binhex40		[MacMime,Patrik Faltstrom]
	news-message-id		[RFC1036, Henry Spencer]
	news-transmission		[RFC1036, Henry Spencer]
	wordperfect5.1		[Paul Lindner]
	pdf		[Paul Lindner]
	zip		[Paul Lindner]
	macwriteii		[Paul Lindner]
	msword		[Paul Lindner]
	remote-printing		[RFC1486,Rose]
	mathematica		[Van Nostern]
	cybercash		[Eastlake]
	commonground		[Glazer]
	iges		[Parks]
	riscos		[Smith]
	eshop		[Katz]
	x400-bp		[RFC1494]
	sgml		[RFC1874]
	cals-1840		[RFC1895]
	pgp-encrypted		[RFC3156]
	pgp-signature		[RFC3156]
	pgp-keys		[RFC3156]
	vnd.framemaker		[Wexler]
	vnd.mif		[Wexler]
	vnd.ms-excel		[Gill]
	vnd.ms-powerpoint		[Gill]
	vnd.ms-project		[Gill]
	vnd.ms-works		[Gill]
	vnd.ms-tnef		[Gill]
	vnd.svd		[Becker]
	vnd.music-niff		[Butler]
	vnd.ms-artgalry		[Slawson]
	vnd.truedoc		[Chase]
	vnd.koan		[Cole]
	vnd.street-stream		[Levitt]
	vnd.fdf		[Zilles]
	set-payment-initiation		[Korver]
	set-payment		[Korver]
	set-registration-initiation		[Korver]
	set-registration		[Korver]
	vnd.seemail		[Webb]
	vnd.businessobjects		[Imoucha]
	vnd.meridian-slingshot		[Wedel]
	vnd.xara		[Matthewman]
	sgml-open-catalog		[Grosso]
	vnd.rapid		[Szekely]
	vnd.enliven		[Santinelli]
	vnd.japannet-registration-wakeup		[Fujii]

vnd.japannet-verification-wakeup [Fujii]
vnd.japannet-payment-wakeup [Fujii]
vnd.japannet-directory-service [Fujii]
vnd.intertrust.digibox [Tomasello]
vnd.intertrust.nncp [Tomasello]
prs.alvestrand.titrax-sheet [Alvestrand]
vnd.noblenet-web [Solomon]
vnd.noblenet-sealer [Solomon]
vnd.noblenet-directory [Solomon]
prs.nprend [Doggett]
vnd.webturbo [Rehem]
hyperstudio [Domino]
vnd.shana.informed.formtemplate [Selzler]
vnd.shana.informed.formdata [Selzler]
vnd.shana.informed.package [Selzler]
vnd.shana.informed.interchange [Selzler]
vnd.$commerce_battelle [Applebaum]
vnd.osa.netdeploy [Klos]
vnd.ibm.MiniPay [Herzberg]
vnd.japannet-jpnstore-wakeup [Yoshitake]
vnd.japannet-setstore-wakeup [Yoshitake]
vnd.japannet-verification [Yoshitake]
vnd.japannet-registration [Yoshitake]
vnd.hp-HPGL [Pentecost]
vnd.hp-PCL [Pentecost]
vnd.hp-PCLXL [Pentecost]
vnd.musician [Adams]
vnd.FloGraphIt [Floersch]
vnd.intercon.formnet [Gurak]
vemmi [RFC2122]
vnd.ms-asf [Fleischman]
vnd.ecdis-update [Buettgenbach]
vnd.powerbuilder6 [Guy]
vnd.powerbuilder6-s [Guy]
vnd.lotus-wordpro [Wattenberger]
vnd.lotus-approach [Wattenberger]
vnd.lotus-1-2-3 [Wattenberger]
vnd.lotus-organizer [Wattenberger]
vnd.lotus-screencam [Wattenberger]
vnd.lotus-freelance [Wattenberger]
vnd.fujitsu.oasys [Togashi]
vnd.fujitsu.oasys2 [Togashi]
vnd.swiftview-ics [Widener]
vnd.dna [Searcy]
prs.cww [Rungchavalnont]
vnd.wt.stf [Wohler]
vnd.dxr [Duffy]
vnd.mitsubishi.misty-guard.trustweb [Tanaka]
vnd.ibm.modcap [Hohensee]
vnd.acucobol [Lubin]
vnd.fujitsu.oasys3 [Okudaira]
marc [RFC2220]
vnd.fujitsu.oasysprs [Ogita]
vnd.fujitsu.oasysgp [Sugimoto]
vnd.visio [Sandal]
vnd.netfpx [Mutz]
vnd.audiograph [Slusanschi]
vnd.epson.salt [Nagatomo]
vnd.3M.Post-it-Notes [O'Brien]
vnd.novadigm.EDX [Swenson]
vnd.novadigm.EXT [Swenson]
vnd.novadigm.EDM [Swenson]
vnd.claymore [Simpson]
vnd.comsocaller [Dellutri]
pkcs7-mime [RFC2311]
pkcs7-signature [RFC2311]
pkcs10 [RFC2311]

vnd.yellowriver-custom-menu [Yellow]
vnd.ecowin.chart [Olsson]
vnd.ecowin.series [Olsson]
vnd.ecowin.filerequest [Olsson]
vnd.ecowin.fileupdate [Olsson]
vnd.ecowin.seriesrequest [Olsson]
vnd.ecowin.seriesupdate [Olsson]
EDIFACT [RFC1767]
EDI-X12 [RFC1767]
EDI-Consent [RFC1767]
vnd.wrq-hp3000-labelled [Bartram]
vnd.minisoft-hp3000-save [Bartram]
vnd.ffsns [Holstage]
vnd.hp-hps [Aubrey]
vnd.fujixerox.docuworks [Taguchi]
xml [RFC3023]
xml-external-parsed-entity [RFC3023]
xml-dtd [RFC3023]
vnd.anser-web-funds-transfer-initiation [Mori]
vnd.anser-web-certificate-issue-initiation [Mori]
vnd.is-xpr [Natarajan]
vnd.intu.qbo [Scratchley]
vnd.publishare-delta-tree [Ben-Kiki]
vnd.cybank [Helmee]
batch-SMTP [RFC2442]
vnd.uplanet.alert [Martin]
vnd.uplanet.cacheop [Martin]
vnd.uplanet.list [Martin]
vnd.uplanet.listcmd [Martin]
vnd.uplanet.channel [Martin]
vnd.uplanet.bearer-choice [Martin]
vnd.uplanet.signal [Martin]
vnd.uplanet.alert-wbxml [Martin]
vnd.uplanet.cacheop-wbxml [Martin]
vnd.uplanet.list-wbxml [Martin]
vnd.uplanet.listcmd-wbxml [Martin]
vnd.uplanet.channel-wbxml [Martin]
vnd.uplanet.bearer-choice-wbxml [Martin]
vnd.epson.quickanime [Gu]
vnd.commonspace [Chandhok]
vnd.fut-misnet [Pruulmann]
vnd.xfdl [Manning]
vnd.intu.qfx [Scratchley]
vnd.epson.ssf [Hoshina]
vnd.epson.msf [Hoshina]
vnd.powerbuilder7 [Shilts]
vnd.powerbuilder7-s [Shilts]
vnd.lotus-notes [Laramie]
pkixcmp [RFC2510]
vnd.wap.wmlc [Stark]
vnd.wap.wmlscriptc [Stark]
vnd.motorola.flexsuite [Patton]
vnd.wap.wbxml [Stark]
vnd.motorola.flexsuite.wem [Patton]
vnd.motorola.flexsuite.kmr [Patton]
vnd.motorola.flexsuite.adsi [Patton]
vnd.motorola.flexsuite.fis [Patton]
vnd.motorola.flexsuite.gotap [Patton]
vnd.motorola.flexsuite.ttc [Patton]
vnd.ufdl [Manning]
vnd.accpac.simply.imp [Leow]
vnd.accpac.simply.aso [Leow]
vnd.vcx [T.Sugimoto]
ipp [RFC2910]
ocsp-request [RFC2560]
ocsp-response [RFC2560]
vnd.previewsystems.box [Smolgovsky]

vnd.mediastation.cdkey [Flurry]
vnd.pg.format [Gandert]
vnd.pg.osasli [Gandert]
vnd.hp-hpid [Gupta]
pkix-cert [RFC2585]
pkix-crl [RFC2585]
vnd.Mobius.TXF [Kabayama]
vnd.Mobius.PLC [Kabayama]
vnd.Mobius.DIS [Kabayama]
vnd.Mobius.DAF [Kabayama]
vnd.Mobius.MSL [Kabayama]
vnd.cups-raster [Sweet]
vnd.cups-postscript [Sweet]
vnd.cups-raw [Sweet]
index [RFC2652]
index.cmd [RFC2652]
index.response [RFC2652]
index.obj [RFC2652]
index.vnd [RFC2652]
vnd.triscape.mxs [Simonoff]
vnd.powerbuilder75 [Shilts]
vnd.powerbuilder75-s [Shilts]
vnd.dpgraph [Parker]
http [RFC2616]
sdp [RFC2327]
vnd.eudora.data [Resnick]
vnd.fujixerox.docuworks.binder [Matsumoto]
vnd.vectorworks [Pharr]
vnd.grafeq [Tupper]
vnd.bmi [Gotoh]
vnd.ericsson.quickcall [Tidwell]
vnd.hzn-3d-crossword [Minnis]
vnd.wap.slc [WAP-Forum]
vnd.wap.sic [WAP-Forum]
vnd.groove-injector [Joseph]
vnd.fujixerox.ddd [Onda]
vnd.groove-account [Joseph]
vnd.groove-identity-message [Joseph]
vnd.groove-tool-message [Joseph]
vnd.groove-tool-template [Joseph]
vnd.groove-vcard [Joseph]
vnd.ctc-posml [Kohlhepp]
vnd.canon-lips [Muto]
vnd.canon-cpdl [Muto]
vnd.trueapp [Hepler]
vnd.s3sms [Tarkkala]
iotp [RFC2935]
vnd.mcd [Gotoh]
vnd.httphone [Lefevre]
vnd.informix-visionary [Gales]
vnd.msign [Borcherding]
vnd.ms-lrm [Ledoux]
vnd.contact.cmsg [Patz]
vnd.epson.esf [Hoshina]
whoispp-query [RFC2957]
whoispp-response [RFC2958]
vnd.mozilla.xul+xml [McDaniel]
parityfec [RFC3009]
vnd.palm [Peacock]
vnd.fsc.weblaunch [D.Smith]
vnd.tve-trigger [Welsh]
dvcs [RFC3029]
sieve [RFC3028]
vnd.vividence.scriptfile [Risher]
vnd.hhe.lesson-player [Jones]
beep+xml [RFC3080]
font-tdpfr [RFC3073]

vnd.mseq [Le Bodic]
vnd.aether.imp [Moskowitz]
vnd.Mobius.MQY [Devasia]
vnd.Mobius.MBK [Devasia]
vnd.vidsoft.vidconference [Hess]
vnd.ibm.afplinedata [Buis]
vnd.irepository.package+xml [Knowles]
vnd.sss-ntf [Bruno]
vnd.sss-dtf [Bruno]
vnd.sss-cod [Dani]
vnd.pvi.ptid1 [Lamb]
isup [RFCISUP]
qsig [RFCISUP]
timestamp-query [RFC3161]
timestamp-reply [RFC3161]

image
jpeg [RFC2045,RFC2046]
gif [RFC2045,RFC2046]
ief Image Exchange Format [RFC1314]
g3fax [RFC1494]
tiff Tag Image File Format [RFC2302]
cgm Computer Graphics Metafile [Francis]
naplps [Ferber]
vnd.dwg [Moline]
vnd.svf [Moline]
vnd.dxf [Moline]
png [Randers-Pehrson]
vnd.fpx [Spencer]
vnd.net-fpx [Spencer]
vnd.xiff [SMartin]
prs.btif [Simon]
vnd.fastbidsheet [Becker]
vnd.wap.wbmp [Stark]
prs.pti [Laun]
vnd.cns.inf2 [McLaughlin]
vnd.mix [Reddy]
vnd.fujixerox.edmics-rlc [Onda]
vnd.fujixerox.edmics-mmr [Onda]
vnd.fst [Fuldseth]

audio
basic [RFC2045,RFC2046]
32kadpcm [RFC2421,RFC2422]
vnd.qcelp [Lundblade]
vnd.digital-winds [Strazds]
vnd.lucent.voice [Vaudreuil]
vnd.octel.sbc [Vaudreuil]
vnd.rhetorex.32kadpcm [Vaudreuil]
vnd.vmx.cvsd [Vaudreuil]
vnd.nortel.vbk [Parsons]
vnd.cns.anp1 [McLaughlin]
vnd.cns.inf1 [McLaughlin]
L16 [RFC2586]
vnd.everad.plj [Cicelsky]
telephone-event [RFC2833]
tone [RFC2833]
prs.sid [Walleij]
vnd.nuera.ecelp4800 [Fox]
vnd.nuera.ecelp7470 [Fox]
mpeg [RFC3003]
parityfec [RFC3009]
MP4A-LATM [RFC3016]
vnd.nuera.ecelp9600 [Fox]
G.722.1 [RFC3047]
mpa-robust [RFC3119]
vnd.cisco.nse [Kumar]

video
mpeg [RFC2045,RFC2046]

quicktime	[Paul Lindner]
vnd.vivo	[Wolfe]
vnd.motorola.video	[McGinty]
vnd.motorola.videop	[McGinty]
vnd.fvt	[Fuldseth]
pointer	[RFC2862]
parityfec	[RFC3009]
vnd.mpegurl	[Recktenwald]
MP4V-ES	[RFC3016]
vnd.nokia.interleaved-multimedia	[Kangaslampi]
model	[RFC2077]
iges	[Parks]
vrml	[RFC2077]
mesh	[RFC2077]
vnd.dwf	[Pratt]
vnd.gtw	[Ozaki]
vnd.flatland.3dml	[Powers]
vnd.vtu	[Rabinovitch]
vnd.mts	[Rabinovitch]
vnd.gdl	[Babits]
vnd.gs-gdl	[Babits]
vnd.parasolid.transmit.text	[Dearnaley,Juckes]
vnd.parasolid.transmit.binary	[Dearnaley,Juckes]

The "media-types" directory contains a subdirectory for each content type and each of those directories contains a file for each content subtype.

```
                              |-application-
                              |-audio-------
                              |-image-------
                 |-media-types-|-message-----
                              |-model-------
                              |-multipart---
                              |-text--------
                              |-video-------
```

URL = ftp://ftp.isi.edu/in-notes/iana/assignments/media-types

Character Sets

All of the character sets listed the section on Character Sets are registered for use with MIME as MIME Character Sets. The correspondance between the few character sets listed in the MIME specifications [RFC2045,RFC2046] and the list in that section are:

Type	Description	Reference
US-ASCII	see ANSI_X3.4-1968 below	[RFC2045,RFC2046]
ISO-8859-1	see ISO_8859-1:1987 below	[RFC2045,RFC2046]
ISO-8859-2	see ISO_8859-2:1987 below	[RFC2045,RFC2046]
ISO-8859-3	see ISO_8859-3:1988 below	[RFC2045,RFC2046]
ISO-8859-4	see ISO_8859-4:1988 below	[RFC2045,RFC2046]
ISO-8859-5	see ISO_8859-5:1988 below	[RFC2045,RFC2046]
ISO-8859-6	see ISO_8859-6:1987 below	[RFC2045,RFC2046]
ISO-8859-7	see ISO_8859-7:1987 below	[RFC2045,RFC2046]
ISO-8859-8	see ISO_8859-8:1988 below	[RFC2045,RFC2046]
ISO-8859-9	see ISO_8859-9:1989 below	[RFC2045,RFC2046]

Access Types

Type	Description	Reference
FTP		[RFC2045,RFC2046]
ANON-FTP		[RFC2045,RFC2046]
TFTP		[RFC2045,RFC2046]
AFS		[RFC2045,RFC2046]
LOCAL-FILE		[RFC2045,RFC2046]
MAIL-SERVER		[RFC2045,RFC2046]
content-id		[RFC1873]

Conversion Values

Conversion values or Content Transfer Encodings.

Type	Description	Reference
7BIT		[RFC2045,RFC2046]
8BIT		[RFC2045,RFC2046]
BASE64		[RFC2045,RFC2046]
BINARY		[RFC2045,RFC2046]
QUOTED-PRINTABLE		[RFC2045,RFC2046]

MIME / X.400 MAPPING TABLES

MIME to X.400 Table

MIME content-type	X.400 Body Part	Reference
text/plain		
charset=us-ascii	ia5-text	[RFC1494]
charset=iso-8859-x	EBP - GeneralText	[RFC1494]
text/richtext	no mapping defined	[RFC1494]
application/oda	EBP - ODA	[RFC1494]
application/octet-stream	bilaterally-defined	[RFC1494]
application/postscript	EBP - mime-postscript-body	[RFC1494]
image/g3fax	g3-facsimile	[RFC1494]
image/jpeg	EBP - mime-jpeg-body	[RFC1494]
image/gif	EBP - mime-gif-body	[RFC1494]
audio/basic	no mapping defined	[RFC1494]
video/mpeg	no mapping defined	[RFC1494]

Abbreviation: EBP - Extended Body Part

X.400 to MIME Table

Basic Body Parts

X.400 Basic Body Part	MIME content-type	Reference
ia5-text	text/plain;charset=us-ascii	[RFC1494]
voice	No Mapping Defined	[RFC1494]
g3-facsimile	image/g3fax	[RFC1494]
g4-class1	no mapping defined	[RFC1494]
teletex	no mapping defined	[RFC1494]
videotex	no mapping defined	[RFC1494]
encrypted	no mapping defined	[RFC1494]
bilaterally-defined	application/octet-stream	[RFC1494]
nationally-defined	no mapping defined	[RFC1494]
externally-defined	See Extended Body Parts	[RFC1494]

X.400 Extended Body Part	MIME content-type	Reference
GeneralText	text/plain;charset=iso-8859-x	[RFC1494]

ODA	application/oda	[RFC1494]
mime-postscript-body	application/postscript	[RFC1494]
mime-jpeg-body	image/jpeg	[RFC1494]
mime-gif-body	image/gif	[RFC1494]

REFERENCES

[MacMime] Work in Progress.

[RFC1036] Horton, M., and R. Adams, "Standard for Interchange of USENET Messages", RFC 1036, AT&T Bell Laboratories, Center for Seismic Studies, December 1987.

[RFC1494] Alvestrand, H., and S. Thompson, "Equivalences between 1988 X.400 and RFC-822 Message Bodies", RFC 1494, SINTEF DELAB, Soft*Switch, Inc., August 1993.

[RFC1563] Borenstien, N., "The text/enriched MIME content-type". RFC 1563, Bellcore, January 1994.

[RFC1767] Crocker, D., "MIME Encapsulation of EDI Objects". RFC 1767, Brandenburg Consulting, March 1995.

[RFC1866] Berners-Lee, T., and D. Connolly, "Hypertext Markup Language - 2.0", RFC 1866, MIT/W3C, November 1995.

[RFC1873] Levinson, E., "Message/External-Body Content-ID Access Type", RFC 1873, Accurate Information Systems, Inc. December 1995.

[RFC1874] Levinson, E., "SGML Media Types", RFC 1874, Accurate Information Systems, Inc. December 1995.

[RFC1892] Vaudreuil, G., "The Multipart/Report Content Type for the Reporting of Mail System Administrative Messages", RFC 1892, Octel Network Services, January 1996.

[RFC1894] Moore, K. and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 1894, University of Tennessee, Octel Network Services, January 1996.

[RFC1895] Levinson, E., "The Application/CALS-1840 Content Type", RFC 1895, Accurate Information Systems, February 1996.

[RFC1896] Resnick, P., and A. Walker, "The Text/Enriched MIME Content Type", RFC 1896, Qualcomm, Intercon, February 1996.

[RFC1945] Berners-Lee, Y., R. Feilding, and H.Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945. MIT/LCS, UC Irvine, MIT/LCS, May 1996.

[RFC2045] Freed, N., and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2046] Freed, N., and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC2077] Nelson, S., C. Parks, and Mitra, "The Model Primary Content Type for Multipurpose Internet Mail Extensions", RFC 2077, LLNL, NIST, WorldMaker, January 1997.

[RFC2122] Mavrakis, D., Layec, H., and K. Kartmann, "VEMMI URL Specification", RFC 2122, Monaco Telematique MC-TEL, ETSI, Telecommunication+Multimedia, March 1997.

[RFC2220] Guenther, R., "The Application/MARC Content-type", RFC 2220, Library of Congress, Network Devt. & MARC Standards Office, October 1997.

[RFC2298] Fajman, R., "An Extensible Message Format for Message Disposition Notifications", RFC 2298, March 1998.

[RFC2302] Parsons, G., et. al., "Tag Image File Format (TIFF) - image/tiff", RFC 2302, March 1998.

[RFC2311] Dusse, S., et. al., "S/MIME Version 2 Message Specification, RFC 2311, March 1998.

[RFC2318] Lie, H., Bos, B., and C. Lilley, "The text/css Media Type", RFC 2318, March 1998.

[RFC2327] Handley, M., and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1999.

[RFC2387] Levinson, E., "The MIME Multipart/Related Content-type", RFC 2387, XIson Inc, August 1998.

[RFC2388] Masinter, L., "Form-based File Upload in HTML", RFC 2388, Xerox Corporation, August 1998.

[RFC2421] Vaudreuil, G., and G. Parsons, "Voice Profile for Internet Mail - version 2", RFC 2421, September 1998.

[RFC2422] Vaudreuil, G., and G. Parsons, "Toll Quality Voice - 32 kbit/s ADPCM MIME Sub-type Registration", RFC 2422, September 1998.

[RFC2423] Vaudreuil, G., and G. Parsons, "VPIM Voice Message MIME Sub-type Registration", RFC 2423, September 1998.

[RFC2425] Howes, T., Smith, M., and F. Dawson, "A MIME Content-Type for Directory Information", RFC 2425, September 1998.

[RFC2442] Freed, N., Newman, D., Belissent, J. and M. Hoy, "The Batch SMTP Media Type", RFC 2442, November 1998.

[RFC2445] Dawson, F., and D. Stenerson, "Internet Calendaring and Scheduling Core Object Specification (iCalendar)", RFC 2445, November 1998.

[RFC2483] M. Mealling and R. Daniel, "URI resolution services necessary for URN resolution", RFC 2483, January 1999.

[RFC2510] Adams, C., and S. Farrell, "Internet X.509 Public Key Infrastructure Certificate Management Protocols", RFC 2510, March 1999.

[RFC2560] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.

[RFC2585] Housley, R. and P. Hoffman, "Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP", RFC 2585, May 1999.

[RFC2586] Salsman, J and H. Alvestrand, "The Audio/L16 MIME content type", RFC 2586, May 1999.

[RFC2616] Fielding, R., et. al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2652] Allen, J., and M. Mealling, "MIME Object Definitions for the

Common Indexing Protocol (CIP)", RFC 2652, August 1999.

[RFC2793] Hellstrom, G., "RTP Payload for Text Conversation", RFC 2793, May 2000.

[RFC2833] Schulzrinne, H., "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals", RFC 2833, May 2000.

[RFC2854] Connolly, D., and L. Masinter, "The 'text/html' Media Type", RFC 2854, June 2000.

[RFC2862] Civanlar, M., and G. Cash, "RTP Payload Format for Real-Time Pointers", RFC 2862, June 2000.

[RFC2910] Herriot, R., Editor, Butler, S., Moore, P., Turner, R. and J. Wenn, "Internet Printing Protocol/1.0: Encoding and Transport", RFC 2910, September 2000.

[RFC2935] Eastlake, D. and C. Smith, "Internet Open Trading Protocol (IOTP) HTTP Supplement", RFC 2935, September 2000.

[RFC3003] M. Nilsson, "The audio/mpeg Media Type", RFC 3003, November 2000.

[RFC3009] J.Rosenberg and H.Schulzrinne, "Registration of parityfec MIME types", RFC 3009, November 2000.

[RFC3016] Kikuchi, Y., T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata, "RTP payload format for MPEG-4 Audio/Visual streams", RFC 3016, November 2000.

[RFC3023] M. Murata, S. St.Laurent, and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3028] T. Showalter, "Sieve: A Mail Filtering Language", RFC 3028, January 2001.

[RFC3029] Adams,C. , P. Sylvester, M. Zolotarev, and R. Zuccherato, "Internet X.509 Public Key Infrastructure Data Validation and Certification Server Protocols", RFC 3029, January 2001.

[RFC3047] Luthi, P. "RTP Payload Format for ITU-T Recommendation G.772.1", RFC 3047, January 2001.

[RFC3073] Collins, J., "Portable Font Resource (PFR) - application/font-tdpfr MIME Sub-type Registration", RFC 3073, February 2001.

[RFC3080] Rose, M., "The Blocks Extensible Exchange Protocol Core", RFC 3080, February 2001.

[RFC3119] R. Finlayson, "A More Loss-Tolerant RTP Payload Format for MP3 Audio", RFC 3119, June 2001.

[RFCISUP] E. Zimmerer, J. Peterson, A. Vemuri, L. Ong, F. Audet, M. Watson, and M. Zonoun, "MIME media types for ISUP and QSIG Objects", RFC XXXX, Month Year.

[RFC3156] M. Elkins, D. Del Torto, R. Levien, and T. Roessler, "MIME Security with OpenPGP", RFC 3156, August 2001.

[RFC3161] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time Stamp Protocol (TSP)", RFC 3161, August 2001.

PEOPLE

[Adams] Greg Adams <gadams@waynesworld.ucsd.edu>, March 1997.

[Allen] Steve Allen <sla@ucolick.org>, September 1997.

[Alvestrand] Harald T. Alvestrand <Harald.T.Alvestrand@uninett.no>, January 1997.

[Applebaum] David Applebaum, <applebau@battelle.org>, February 1997.

[Aubrey] Steve Aubrey, <steve_aubrey@hp.com>, July 1998.

[Babits] Attila Babits, <ababits@graphisoft.hu>, April 2000, May 2000.

[Bartram] Chris Bartram, <RCB@3k.com>, May 1998.

[Becker] Scott Becker, <scottb@bxwa.com>, April 1996, October 1998.

[Ben-Kiki] Oren Ben-Kiki, <oren@capella.co.il>, October 1998.

[Berners-Lee] Tim Berners-Lee, <timbl@w3.org>, May 1996.

[Borcherding] Malte Borcherding, <Malte.Borcherding@brokat.com>, August 2000.

[Borenstein] Nathaniel Borenstein, <NSB@bellcore.com>, April 1994.

[Bradley] Dan Bradley, <dan@dantom.com>, October 2000.

[Bruno] Eric Bruno, <ebruno@solution-soft.com>, June 2001.

[Buettgenbach] Gert Buettgenbach, <bue@sevencs.com>, May 1997.

[Buis] Roger Buis, <buis@us.ibm.com>, March 2001.

[Butler] Tim Butler, <tim@its.bldrdoc.gov>, April 1996.

[Larry Campbell]

[Chandhok] Ravinder Chandhok, <chandhok@within.com>, December 1998.

[Chase] Brad Chase, <brad_chase@bitstream.com>, May 1996.

[Cicelsky] Shay Cicelsky, <shayc@everad.com>, May 2000.

[Cole] Pete Cole, <pcole@sseyod.demon.co.uk>, June 1996.

[Dave Crocker] Dave Crocker <dcrocker@mordor.stanford.edu>

[Terry Crowley]

[Dani] Asang Dani, <adani@solution-soft.com>, June 2001.

[Daniel] Ron Daniel, Jr. <rdaniel@lanl.gov>, June 1997.

[Dearnaley] Roger Dearnaley, <x_dearna@ugsolutions.com>, October 2000.

[Dellutri] Steve Dellutri, <sdellutri@cosmocom.com>, March 1998.

[Devasia] Alex Devasia, <adevasia@mobius.com>, March 2001.

[Doggett] Jay Doggett, <jdoggett@tiac.net>, February 1997.

[Domino] Michael Domino, <michaeldomino@mediaone.net>, February 1997.

[Duffy] Michael Duffy, <miked@psiaustin.com>, September 1997.

[Eastlake] Donald E. Eastlake 3rd, <Donald.Eastlake@motorola.com>, April 1995, May 2000.

[Faltstrom] Patrik Faltstrom <paf@nada.kth.se>

[Fleischman] Eric Fleischman <ericfl@MICROSOFT.com>, April 1997.

[Floersch] Dick Floersch <floersch@echo.sound.net>, March 1997.

[Flurry] Henry Flurry <henryf@mediastation.com>, April 1999.

[Fox] Michael Fox, <mfox@nuera.com>, August 2000, January 2001.

[Francis] Alan Francis, A.H.Francis@open.ac.uk, December 1995.

[Fujii] Kiyofusa Fujii <kfujii@japannet.or.jp>, February 1997.

[Fuldseth] Arild Fuldseth, <Arild.Fuldseth@fast.no>, June 2000.

[Gales] Christopher Gales, <christopher.gales@informix.com>, August 2000.

[Gandert] April Gandert <gandert.am@pg.com>, April 1999.

[Gill] Sukvinder S. Gill, <sukvg@microsoft.com>, April 1996.

[Glazer] David Glazer, <dglazer@best.com>, April 1995.

[Gotoh] Tadashi Gotoh, <tgotoh@cadamsystems.co.jp>, February 2000, June 2000.

[Gu] Yu Gu, <guyu@rd.oda.epson.co.jp>, December 1998.

[Gupta] Aloke Gupta <Aloke_Gupta@ex.cv.hp.com>, April 1999.

[Gurak] Tom Gurak, <assoc@intercon.roc.servtech.com>, March 1997.

[Gurney] John-Mark Gurney <jmg@flyidea.com>, August 1999.

[Guy] David Guy, <dguy@powersoft.com>, June 1997.

[Helmee] Nor Helmee, <helmee@my.cybank.net>, November 1998.

[Hepler] J. Scott Hepler, <scott@truebasic.com>, May 2000.

[Herzberg] Amir Herzberg, <amirh@haifa.vnet.ibm.com>, February 1997.

[Hess] Robert Hess, <hess@vidsoft.de>, March 2001.

[Hodge] Tim Hodge, <thodge@curl.com>, August 2000.

[Hohensee] Reinhard Hohensee <rhohensee@VNET.IBM.COM>, September 1997.

[Holstage] Mary Holstage <holstege@firstfloor.com>, May 1998.

[Hoshina] Shoji Hoshina <Hoshina.Shoji@exc.epson.co.jp>, January 1999, September 2000.

[Hurtta] Kari E. Hurtta <flexstor@ozone.FMI.FI>

[Imoucha] Philippe Imoucha <pimoucha@businessobjects.com>, October 1996.

[IPTC] International Press Telecommunications Council (David Allen), <m_director_iptc@dial.pipex.com>, July 2000.

[Jones] Randy Jones, <randy_jones@archipelago.com>, January 2001.

[Joseph] Todd Joseph <todd_joseph@groove.net>, February 2000, March 2000, April 2000.

[Juckes] John Juckes, <johnj@ugsolutions.com>, October 2000.

[Kangaslampi] Petteri Kangaslampi, <petteri.kangaslampi@nokia.com>, March 2001.

[Katz] Steve Katz, <skatz@eshop.com>, June 1995.

[Klos] Steven Klos, <stevek@osa.com>, February 1997.

[Knowles] Martin Knowles, <mjk@irepository.net>, June 2001.

[Kohlhepp] Bayard Kohlhepp, <bayard@ctcexchange.com>, April 2000.

[Korver] Brian Korver <briank@terisa.com>, October 1996.

[Kumar] Rajesh Kumar, <rkumar@cisco.com>, August 2001.

[Lamb] Charles P. Lamb, <CLamb@pvimage.com>, June 2001.

[Laramie] Michael Laramie <laramiem@btv.ibm.com>, February 1999.

[Laun] Juern Laun <juern.laun@gmx.de>, April 1999.

[Le Bodic] Gwenael Le Bodic <Gwenael.Le_Bodic@alcatel.fr>, March 2001.

[Ledoux] Eric Ledoux, <ericle@microsoft.com>, August 2000.

[Lefevre] Franck Lefevre, <franck@k1info.com>, August 2000.

[Leow] Steve Leow <Leost01@accpac.com>, April 1999.

[Levitt] Glenn Levitt <streetd1@ix.netcom.com>, October 1996.

[Lines] John Lines <john@paladin.demon.co.uk>, January 1998.

[Lubin] Dovid Lubin <dovid@acucobol.com>, October 1997.

[Lubos] Mikusiak Lubos <lmikusia@blava-s.bratisla.ingr.com>, October 1996.

[Lundblade] Laurence Lundblade <lgl@qualcomm.com>, October 1996.

[Manning] Dave Manning <dmanning@uwi.com>, January, March 1999.

[Martin] Bruce Martin <iana-registrar@uplanet.com>, November 1998.

[Martin] Steven Martin <smartin@xis.xerox.com>, October 1997.

[Matsumoto] Takashi Matsumoto <takashi.matsumoto@fujixerox.co.jp>, February 2000

[Matthewman] David Matthewman <david@xara.com>, October 1996.

[McDaniel] Braden N. McDaniel, <braden@endoframe.com>, October 2000.

[McGinty] Tom McGinty <tmcginty@dma.isg.mot.com>

[McLaughlin] Ann McLaughlin <amclaughlin@comversens.com>, April 1999.

[Minnis] James Minnis <james-minnis@glimpse-of-tomorrow.com>, February 2000

[Moline] Jodi Moline, <jodim@softsource.com>, April 1996.

[Mori] Hiroyoshi Mori, <mori@mm.rd.nttdata.co.jp>, August 1998.

[Moskowitz] Jay Moskowitz, <jay@aethersystems.com>, March 2001.

[Muto] Shin Muto, <shinmuto@pure.cpdc.canon.co.jp>, May 2000.

[Mutz] Andy Mutz, <andy_mutz@hp.com>, December 1997.

[Nagatomo] Yasuhito Nagatomo <naga@rd.oda.epson.co.jp>, January 1998.

[Natarajan] Satish Natarajan, <satish@infoseek.com>, August 1998.

[Nelson] Jan Nelson, <jann@microsoft.com>, May 2000.

[Nilsson] Martin Nilsson, <nilsson@id3.org>, October 2000.

[O'Brien] Michael O'Brien <meobrien1@mmm.com>, January 1998.

[Ogita] Masumi Ogita, <ogita@oa.tfl.fujitsu.co.jp>, October 1997.

[Okudaira] Seiji Okudaira <okudaira@candy.paso.fujitsu.co.jp>, October 1997.

[Olsson] Thomas Olsson <thomas@vinga.se>, April 1998.

[Onda] Masanori Onda <Masanori.Onda@fujixerox.co.jp>, February 2000.

[Ozaki] Yutaka Ozaki <yutaka_ozaki@gen.co.jp>, January 1999.

[Paul Lindner]

[Parker] David Parker <davidparker@davidparker.com>, August 1999.

[Parks] Curtis Parks, <parks@eeel.nist.gov>, April 1995.

[Parsons] Glenn Parsons <gparsons@nortelnetworks.com>, February 1999.

[Patton] Mark Patton <fmp014@email.mot.com>, March 1999.

[Patz] Frank Patz, <fp@contact.de>, September 2000.

[Peacock] Gavin Peacock, <gpeacock@palm.com>, November 2000.

[Pentecost] Bob Pentecost, <bpenteco@boi.hp.com>, March 1997.

[Pharr] Paul C. Pharr <pharr@diehlgraphsoft.com>, February 2000.

[Powers] Michael Powers, <powers@insideout.net>, January 1998.
<pow@flatland.com>, January 1999.

[Pratt] Jason Pratt, <jason.pratt@autodesk.com>, August 1997.

[Pruulmann] Jann Pruulmann, <jaan@fut.ee>, December 1998.

[Rabinovitch] Boris Rabinovitch <boris@virtue3d.com>, February 2000.

[Randers-Pehrson] Glenn Randers-Pehrson <glennrp@ARL.MIL>, October 1996.

[Recktenwald] Heiko Recktenwald, <uzs106@uni-bonn.de>, November 2000.

[Reddy] Saveen Reddy <saveenr@miscrosoft.com>, July 1999.

[Rehem] Yaser Rehem, <yrehem@sapient.com>, February 1997.

[Resnick] Pete Resnick, <presnick@qualcomm.com>, February 2000.

[Risher] Mark Risher, <markr@vividence.com>, December 2000.

[Rose] Marshall Rose, <mrose@dbc.mtview.ca.us>, April 1995.

[Rosenberg] Jonathan Rosenberg, <jdrosen@dynamicsoft.com>, October 2000.

[Rungchavalnont] Khemchart Rungchavalnont,
<khemcr@cpu.cp.eng.chula.ac.th>, July 1997.

[Sandal] Troy Sandal <troys@visio.com>, November 1997.

[Santinelli] Paul Santinelli, Jr. <psantinelli@narrative.com>, October 1996.

[Scrathcley] Greg Scratchley <greg_scratchley@intuit.com>, October 1998.

[Searcy] Meredith Searcy, <msearcy@newmoon.com>, June 1997.

[Shapiro] Ehud Shapiro

[Shilts] Reed Shilts <reed.shilts@sybase.com>, February 1999, August 1999.

[Simon] Ben Simon, <BenS@crt.com>, September 1998.

[Simonoff] Steven Simonoff <scs@triscape.com>, August 1999.

[Simpson] Ray Simpson <ray@cnation.com>, January 1998.

[Slawson] Dean Slawson, <deansl@microsoft.com>, May 1996.

[Slusanschi] Horia Cristian Slusanschi <H.C.Slusanschi@massey.ac.nz>,
January 1998.

[D.Smith] Derek Smith, <derek@friendlysoftware.com>, November 2000.

[Smith] Nick Smith, <nas@ant.co.uk>, June 1995.

[Smolgovsky] Roman Smolgovsky <romans@previewsystems.com>, April 1999.

[Solomon] Monty Solomon, <monty@noblenet.com>, February 1997.

[Spencer] Marc Douglas Spencer <marcs@itc.kodak.com>, October 1996.

[Henry Spencer]

[Stark] Peter Stark <stark@uplanet.com>, March 1999.

[Strazds] Armands Strazds <armands.strazds@medienhaus-bremen.de>,
January 1999.

[Sugimoto] Masahiko Sugimoto <sugimoto@sz.sel.fujitsu.co.jp>, October 1997.

[T.Sugimoto] Taisuke Sugimoto <sugimototi@noanet.nttdata.co.jp> April 1999.

[Sweet] Michael Sweet <mike@easysw.com>, July 1999.

[Swenson] Janine Swenson <janine@novadigm.com>, January 1998.

[Szekely] Etay Szekely <etay@emultek.co.il>, October 1996.

[Taguchi] Yasuo Taguchi <yasuo.taguchi@fujixerox.co.jp>, July 1998.

[Tanaka] Manabu Tanaka <mtana@iss.isl.melco.co.jp>, September 1997.

[Tarkkala] Lauri Tarkkala, <Lauri.Tarkkala@sonera.com>, May 2000.

[Tidwell] Paul Tidwell <paul.tidwell@ericsson.com>, February 2000.

[Togashi] Nobukazu Togashi <togashi@ai.cs.fujitsu.co.jp>, June 1997.

[Tomasello] Luke Tomasello <luket@intertrust.com>

[Tupper] Jeff Tupper <tupper@peda.com>, February 2000.

[Vaudreuil] Greg Vaudreuil <gregv@lucent.com>, January 1999.

[Walleij] Linus Walleij, <triad@df.lth.se>, July 2000.

[WAP-Forum] WAP Forum Ltd. <wap-feedback@mail.wapforum.org>, February 2000.

[Wattenberger] Paul Wattenberger <Paul_Wattenberger@lotus.com>, June 1997.

[Webb] Steve Webb <steve@wynde.com>, October 1996.

[Wedel] Eric Wedel <ewedel@meridian-data.com>, October 1996.

[Welsh] Linda Welsh, <linda@intel.com>, November 2000.

[Wexler] Mike Wexler, <mwexler@frame.com>, April 1996.

[Widener] Glenn Widener <glennw@ndg.com>, June 1997.

[Wohler] Bill Wohler, <wohler@newt.com>, July 1997.

[Wolfe] John Wolfe, <John_Wolfe.VIVO@vivo.com>, April 1996.

[Van Nostern] Gene C. Van Nostern <gene@wri.com>, February 1995.

[Yellow] Mr. Yellow <yellowriversw@yahoo.com>, March 1998.

[Yoshitake] Jun Yoshitake, <yositake@iss.isl.melco.co.jp>, February 1997.

[Zilles] Steve Zilles <szilles@adobe.com>, October 1996.

[]