QuickTime File Format Specification, May 1996

Developer Press

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

LIMITED WARRANTY ON MEDIA AND REPLACEMENT

If you discover physical defects in the manual or in the media on which a software product is distributed, ADC will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to ADC.

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Figures and Tables

Chapter 1 QuickTime File Format 1

About the QuickTime File Format

This book describes the format and content of QuickTime files. It is intended for developers who need to work with QuickTime files outside the context of the QuickTime environment. For example, if you are developing a non-QuickTime application that imports QuickTime files, you need to understand the material in this book. On the other hand, if you are using QuickTime on any of its supported platforms, you do not necessarily need to be familiar with the file format information presented here.

This book describes QuickTime files in general, rather than how they are supported on a specific computing platform or in a specific programming language. As a result, the file format information is presented in a tabular manner, rather than in coded data structures. Similarly, field names are presented in English rather than as programming language tags. Furthermore, to the extent possible, data types are described generically. For example, this book uses "32-bit signed integer" rather than "`long`" to define a 32-bit integer value. Based on the information provided here, you should be able to create appropriate data structure specifications for your environment.

Finally, QuickTime files are used to store QuickTime movies, as well as other time-based data. If you are writing an application that parses QuickTime files, you should recognize that there may be non-movie data in the files.

For More Information

The *Apple Developer Catalog* (ADC) is Apple Computer's worldwide source for hundreds of development tools, technical resources, training products, and information for anyone interested in developing applications on Apple computer platforms. Customers receive the *Apple Developer Catalog* featuring all current versions of Apple development tools and the most popular third-party development tools. ADC offers convenient payment and shipping options, including site licensing.

To order products or to request a complimentary copy of the *Apple Developer Catalog*, contact

Apple Developer Catalog
Apple Computer, Inc.
P.O. Box 319
Buffalo, NY 14207-0319

Telephone	1-800-282-2732 (United States) 1-800-637-0029 (Canada) 716-871-6555 (International)
Fax	716-871-6511
AppleLink	ORDER.ADC
Internet	order.adc@applelink.apple.com

QuickTime File Format

This document describes how QuickTime movies are stored on disk. The QuickTime file format is designed to accommodate the varied kinds of data that need to be stored in order to work with digital media. Because the file format can be used to describe almost any media structure, it is an ideal format for the exchange of digital media between applications, regardless of the platform on which the application may be running.

This document assumes that the reader is familiar with the basic concepts of digital video and digital audio, as well as with QuickTime. For a complete description of QuickTime concepts, including time coordinate systems and how spatial tracks are composited together, please refer to *Inside Macintosh: QuickTime*.

A QuickTime file stores the description of the media separately from the media data. The description, or meta-data, is called the **movie** and contains information such as the number of tracks, video compression format, and timing information. The movie also contains an index of where all the media data is stored. The media data is all of the actual sample data, such as video frames and audio samples. The media data may be stored in the same file as the QuickTime movie, in a separate file, or in several files.

Before explaining the specifics of how a QuickTime movie is stored, it is important to first understand the basic units that are used to construct QuickTime files. QuickTime uses two basic structures for storing information: **atoms** and **QT atoms**. Both atoms and QT atoms allow you to construct arbitrarily complex hierarchical data structures. Both also allow applications to ignore data they don't understand.

Atom types are specified by a four-character code. Apple Computer reserves all four-character codes consisting entirely of lower case letters.

Unless otherwise stated, all data in a QuickTime movie is stored in big-endian (Motorola) byte ordering.

Finally, all version fields must be set to 0, unless this document states otherwise.

Atoms

The basic data unit in a QuickTime file is the atom**.** Each atom contains size and type information along with its data. The `size` field indicates the number of bytes in the atom, including the `size` and `type` fields. The `type` field specifies the type of data stored in the atom and, by implication, the format of that data. Both the `size` and `type` fields are 32-bit integers.

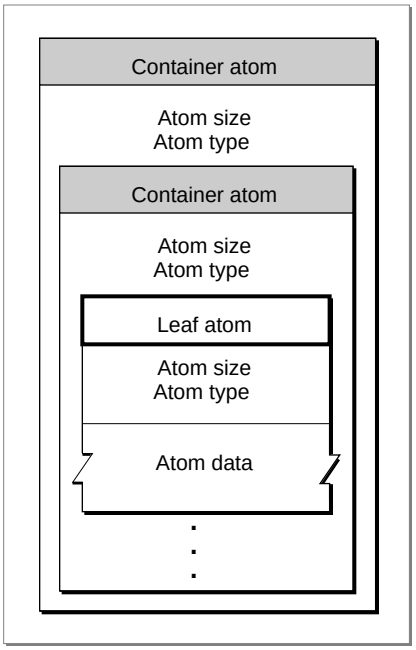
Atoms are recursive in nature. That is, one atom may contain one or more other atoms of varying type. For example, a movie atom contains one track atom for each track in the movie. The track atoms, in turn, contain one or more media atoms each, along with other atoms that define other track and movie characteristics. This hierarchical structure of atoms is referred to as a containment hierarchy.

The format of the data stored within a given atom cannot be determined based only on the `type` field of that atom. That is, an atom's use is determined by its context. A given atom type may have different usages when stored within atoms of different types. This means that all QuickTime file readers must take into consideration not only the atom type, but the atom's containment hierarchy.

Figure 1-1 shows the layout of a sample QuickTime atom. Each atom carries its own size and type information as well as its data. Throughout this chapter, the name of a **container atom** (an atom that contains other atoms, including other container atoms) is printed across a horizontal gray band, and the name of a **leaf atom** (an atom that contains no other atoms) is printed across a horizontal drop shadow box. Leaf atoms contain data, usually in the form of tables.

Atoms within container atoms do not have to be in any particular order, with the exception of handler description atoms. Handler description atoms must come before their data. For example, a media handler description atom must come before a media information atom. A data handler description atom must come before a data information atom.

Figure 1-1 A sample QuickTime atom



Atoms consist of a header, followed by atom data. An atom header consists of the following fields.

Field descriptions

Atom size — A 32-bit integer that indicates the size of the atom, including both the atom header and the atom's contents. If the atom is a leaf atom, then this field contains the size of the single atom. The size of a container atom includes all of its contained atoms.

Type — A 32-bit integer that contains the type of the atom.

The only way to interpret the atom's data is by knowing the type of data that is stored in an atom of a particular type.

QT Atoms

Given the limitations of the structure of the simple atom, Apple has created a new, enhanced data structure called a QT atom. QT atoms provide a more general purpose storage format and remove some of the ambiguities that arise when using simple atoms.

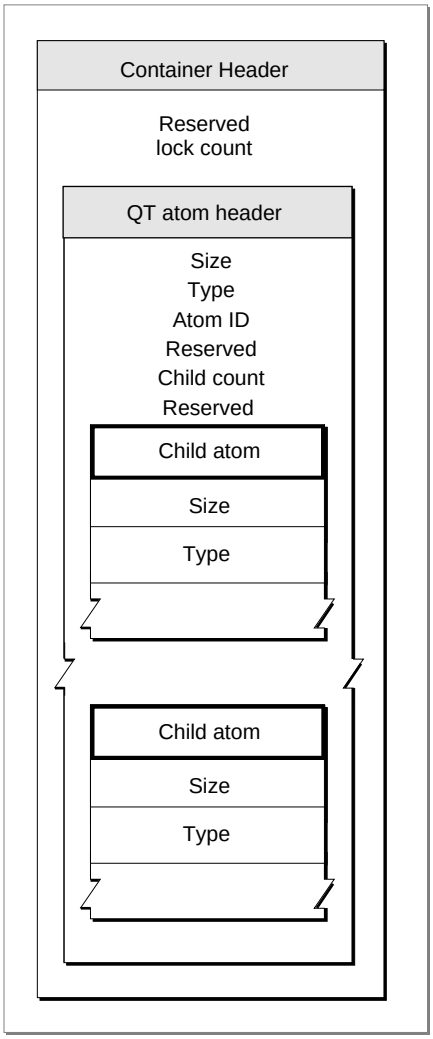
In particular, with simple atoms there is no way to know if an atom is a leaf node or whether it contains other atoms, or both, without specific knowledge about the atom. Using QT atoms, a given node is either is a leaf node or a container node. There is no ambiguity. Furthermore, QT atoms allow for multiple atoms of a given type to be specified through identification numbers. While QT atoms are a more powerful data structure, they require more overhead in the file.

The QuickTime file format uses both atoms and QT atoms. In general, newer parts of the QuickTime file format use QT atoms, and older parts use atoms. When defining new QuickTime structures, you should use QT atoms whenever practical.

Figure 1-2 depicts the layout of a QT atom. Each QT atom starts with a QT atom container header, followed by the root atom. The root atom's type is determined by the QT atom's type. The root atom contains any other atoms that are part of the structure.

Each container atom starts with a QT atom header followed by the atom's contents. The contents are either child atoms or data, but never both. If an atom contains children it also contains all of its children's data and their descendants. The root atom is always present and never has any siblings.

Figure 1-2 QT atom layout



A QT atom container header contains the following data.

Field descriptions

Reserved	A 10-byte element that must be set to 0.
Lock count	A 16-bit integer that must be set to 0.

Each QT atom header contains the following data.

Field descriptions

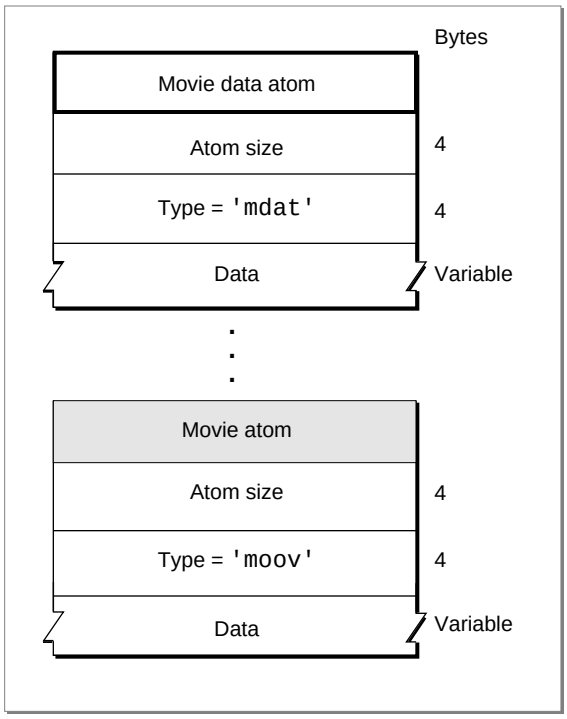
Size	A 32-bit integer that indicates the size of the atom in bytes, including both the QT atom header and the atom's contents. If the atom is a leaf atom, then this field contains the size of the single atom. The size of container atoms includes all of the contained atoms. You can walk the atom tree using the Size and Child count fields (described below).
Type	A 32-bit integer that contains the type of the atom. If this is the root atom, the type value is set to `'sean'`.
Atom ID	A 32-bit integer that contains the atom's ID value. This value must be unique among its siblings. The root atom always has an Atom ID value of 1.
Reserved	A 16-bit integer that must be set to 0.
Child count	A 16-bit integer that specifies the number of child atoms that an atom contains. This count only includes immediate children. If this field is set to 0, the atom is a leaf atom and only contains data.
Reserved	A 32-bit integer that must be set to 0.

File Format

A QuickTime file is simply a collection of atoms. QuickTime does not impose any rules about the order of these atoms.

Figure 1-3 depicts a typical QuickTime file.

Figure 1-3 The structure of a QuickTime file



In file systems that support file name extensions, QuickTime file names typically have an extension of `".mov"`. On the Macintosh platform, QuickTime files have a file type of `"MooV"`. On the Macintosh, the movie atom may be stored as a Macintosh resource using the Resource Manager. The resource has a type of `'moov'`. All media data is stored in the data fork.

As previously discussed, QuickTime files consist of atoms, each with an appropriate atom type. A few of these types are considered basic atom types and form the structure within which the other atoms are stored. Table 1-1 lists the currently supported basic atom types.

Table 1-1 QuickTime file basic atom types

Atom type	Use
`'free'`	Unused space available in file
`'skip'`	Unused space available in file
`'mdat'`	Movie data—usually this data can only be interpreted by using the movie resource
`'pnot'`	Reference to movie preview data
`'moov'`	Movie resource

While it is true that QuickTime imposes no strict order on a movie's atoms, it is often convenient if the movie atom appears near the front of the file. For example, an application that plays a movie over a network would not necessarily have access to the entire movie at all times. If the movie atom is stored at the beginning of the file, the application can use the meta-data to understand the movie's content as it is acquired over the network.

The following sections describe each of these basic atom types in more detail, including descriptions of the atoms that each basic atom may contain.

Free Space Atoms

Both free and skip atoms designate unused space in the movie data file. These atoms consist only of an atom header (atom size and type fields), followed by the appropriate number of bytes of free space. When reading a QuickTime movie, your application may safely skip these atoms. When writing or updating a movie, you may re-use the space associated with these atom types.

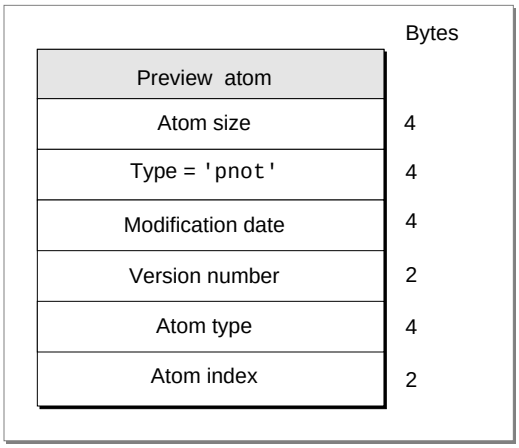
Movie Data Atoms

As with the free and skip atoms, the movie data atom is structured quite simply. It consists of an atom header (atom size and type fields), followed by the movie's media data. Your application can understand the data in this atom only by using the meta-data stored in the movie atom.

Preview Atoms

The preview atom contains information that allows you to find the preview image associated with a QuickTime movie. The preview image, or poster, is a representative image suitable for display to the user in, say, file-open dialogs. Figure 1-4 depicts the layout of a preview atom.

Figure 1-4 The layout of a preview atom



The preview atom has an atom type value of `'pnot'` and, following its atom header, contains the following fields.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this preview atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'pnot'`.
Modification date	A 32-bit unsigned integer containing a date that indicates when the preview was last updated. The date is in standard Macintosh format.
Version number	A 16-bit integer that must be set to 0.
Atom type	A 32-bit integer that indicates the type of atom that contains the preview data. Typically, this is set to `'PICT'` to indicate a QuickDraw picture.
Atom index	A 16-bit integer that identifies which atom of the specified type is to be used as the preview. Typically, this field is set to 1 to indicate that you should use the first atom of the type specified in the Atom type field.

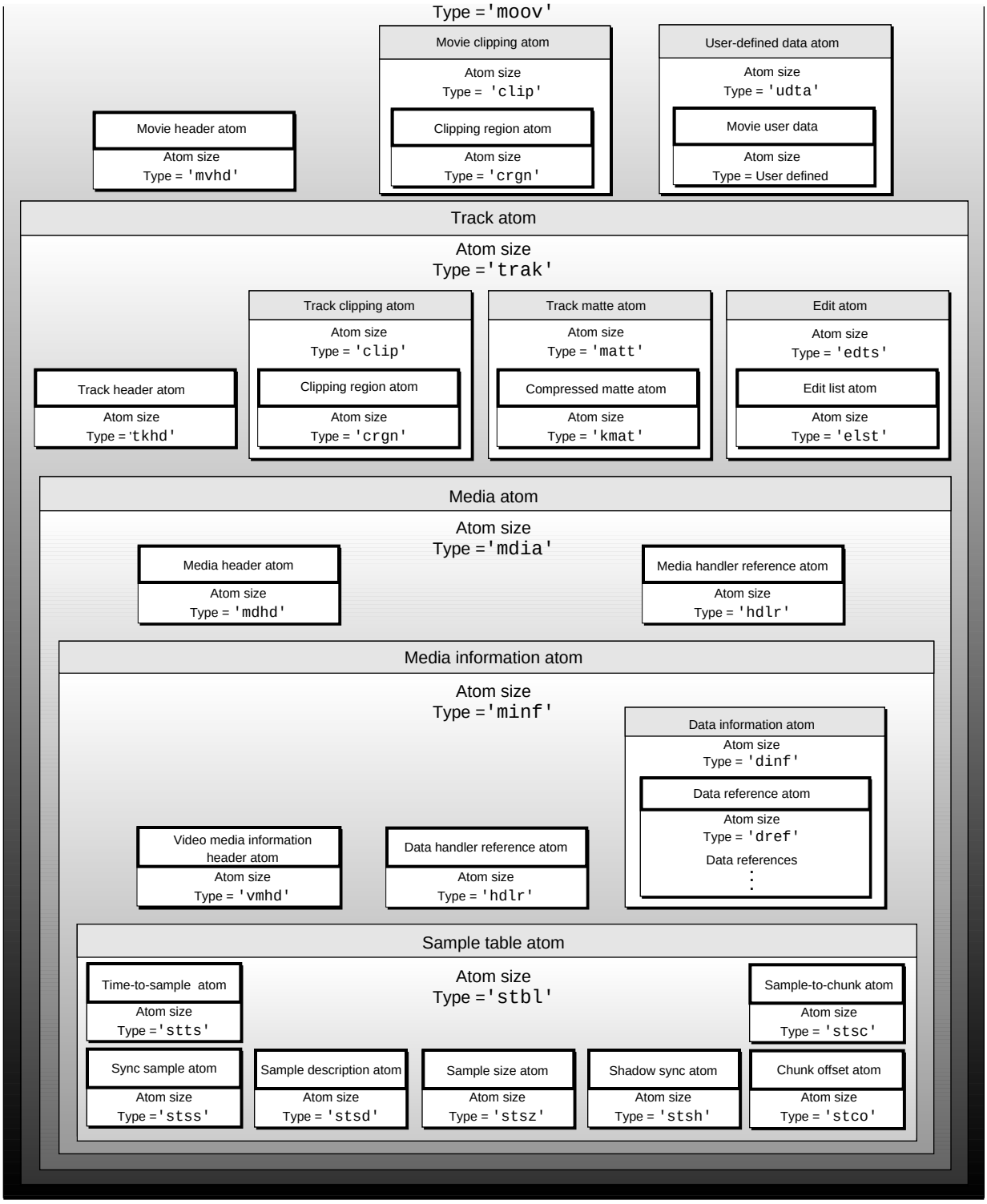
Movie Atoms

Movie atoms have an atom type of `'moov'`. These atoms act as a container for the information that describes a movie's data. This information, or meta-data, is stored in a number of different types of atoms. As such, the movie atom is essentially a container of other atoms. At the highest level, movie atoms contain track atoms, which in turn contain media atoms. At the lowest level you find the leaf atoms, which contain the actual data, usually in the form of a table or a data stream.

For example, the track atom contains the edit atom, which contains a leaf atom called the edit list atom. The edit list atom contains an edit list table. Both of these atoms are discussed later in this book.

Figure 1-5 provides a conceptual view of the organization of a simple, one-track QuickTime movie. Each nested box in the illustration represents an atom that belongs to its containing atom. The figure does not show the data regions of any of the atoms. These areas are described in the sections that follow.

Figure 1-5 Sample organization of a one-track video movie atom



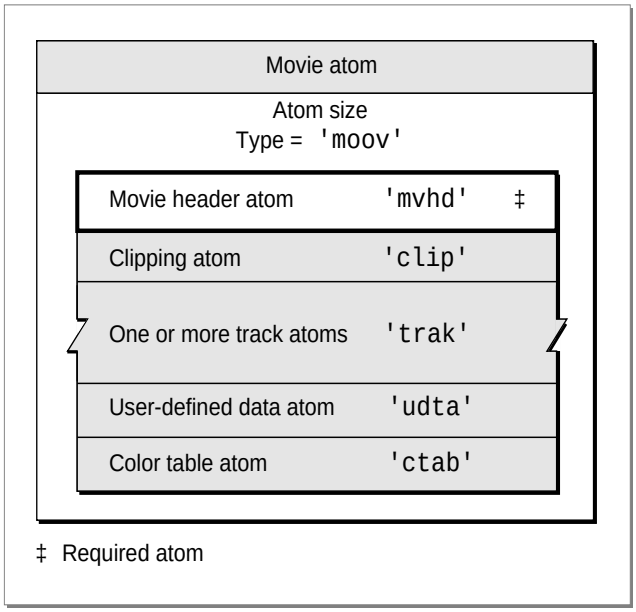
The Movie Atom

You use movie atoms to specify the information that defines a movie—that is, the information that allows your application to understand the data that is stored in the movie data atom. The movie atom contains the movie header atom, which defines the time scale and duration information for the entire movie, as well as its display characteristics. In addition, the movie atom contains each track in the movie.

The movie atom has an atom type of `'moov'`. It contains other types of atoms, including one leaf atom—the movie header (`'mvhd'`)—and several atoms that contain other atoms: a clipping atom (`'clip'`), one or more track atoms (`'trak'`), a color table atom (`'ctab'`), and user data (`'udta'`).

Figure 1-6 shows the layout of a movie atom. The movie header atom is the only required atom in the movie atom.

Figure 1-6 The layout of a movie atom



A movie atom contains the following information.

Field descriptions

Size — The number of bytes in this movie atom.

Type — The type of this movie atom; this field must be set to `'moov'`.

Movie header — The movie header atom associated with this movie. See the next section for details on the movie header atom.

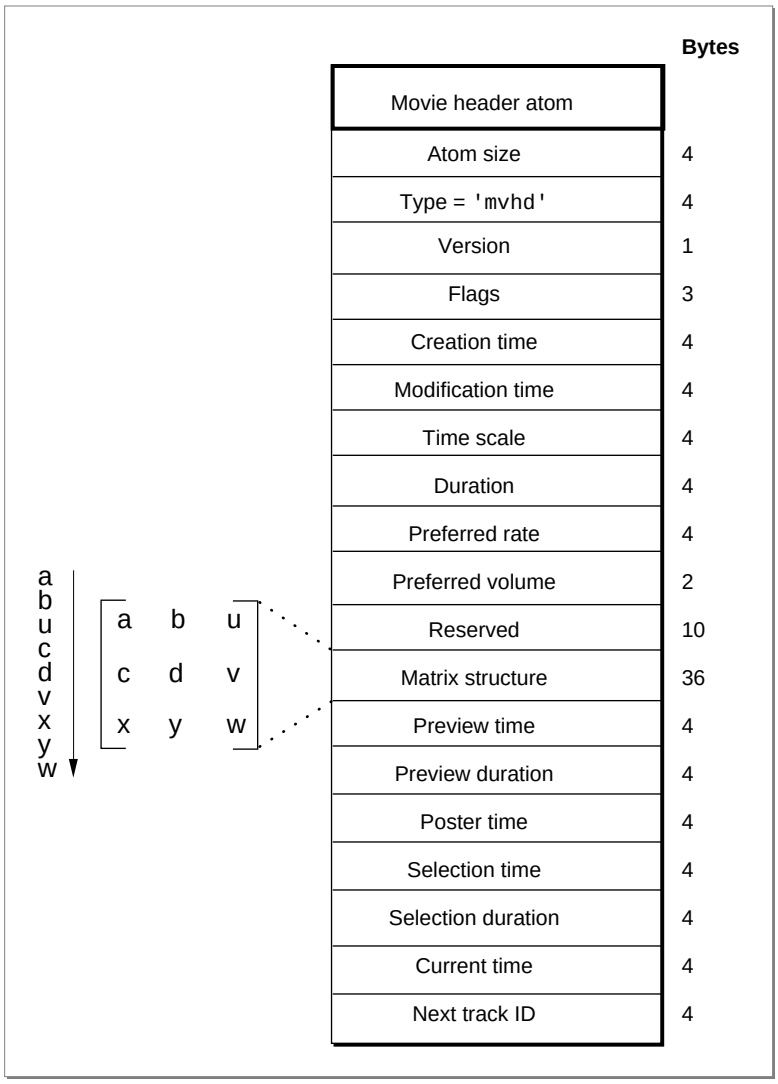
Movie clipping atom — The clipping atom associated with this movie. See “Clipping Atoms,” beginning on page 19, for more information.

Track list	One or more track atoms associated with this movie. See “Track Atoms,” beginning on page 15, for details on track atoms and their associated atoms.
User data	The user data atom associated with this movie. See “User Data Atoms” on page 57 for more information about user data atoms.
Color table	The color table atom associated with this movie. See “Color Table Atoms” on page 14 for a discussion of the color table atom.

Movie Header Atoms

You use the movie header atom to specify the characteristics of an entire QuickTime movie. The data contained in this atom defines characteristics of the entire QuickTime movie, such as time scale and duration. It has an atom type value of `'mvhd'`. Figure 1-7 shows the layout of a movie header atom. The movie header atom is a leaf atom.

Figure 1-7 The layout of a movie header atom



You define a movie header atom by specifying the following data elements.

Field descriptions

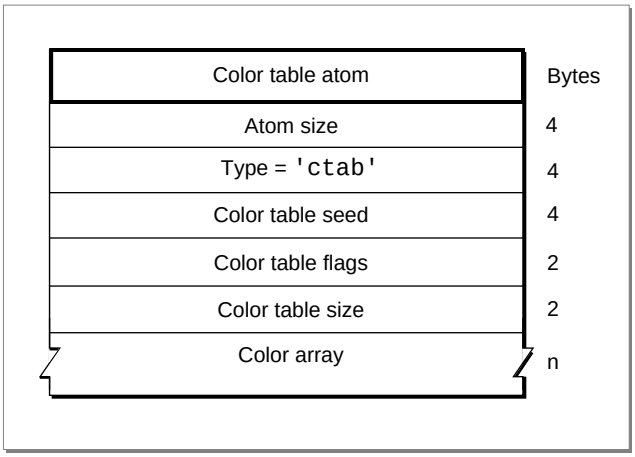
Size	A 32-bit integer that specifies the number of bytes in this movie header atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'mvhd'`.
Version	A 1-byte specification of the version of this movie header atom.
Flags	Three bytes of space for future movie header flags.
Creation time	A 32-bit integer that specifies (in seconds since midnight, January 1, 1904) when the movie atom was created.

Modification time	A 32-bit integer that specifies (in seconds since midnight, January 1, 1904) when the movie atom was changed.
Time scale	A time value that indicates the **time scale** for this movie—that is, the number of time units that pass per second in its time coordinate system. A time coordinate system that measures time in sixtieths of a second, for example, has a time scale of 60.
Duration	A time value that indicates the duration of the movie in time scale units. Note that this property is derived from the movie's tracks. The value of this field corresponds to the duration of the longest track in the movie.
Preferred rate	A 32-bit fixed-point number that specifies the rate at which to play this movie. A value of 1.0 indicates normal rate.
Preferred volume	A 16-bit fixed-point number that specifies how loud to play this movie's sound. A value of 1.0 indicates full volume.
Reserved	Ten bytes reserved for use by Apple. Set to 0.
Matrix	The matrix structure associated with this movie. A matrix shows how to map points from one coordinate space into another.
Preview time	The time value in the movie at which the preview begins.
Preview duration	The duration of the movie preview in movie time scale units.
Poster time	The time value of the time of the movie poster.
Selection time	The time value for the start time of the current selection.
Selection duration	The duration of the current selection in movie time scale units.
Current time	The time value for current time position within the movie.
Next track ID	A 32-bit integer that indicates a value to use for the track ID number of the next track added to this movie. Note that 0 is not a valid track ID value.

Color Table Atoms

Color table atoms define a list of preferred colors for displaying the movie on devices that only support 256 colors. The list may contain up to 256 colors. These optional atoms have a type value of `'ctab'`. The color table atom contains a Macintosh Color Table data structure. Figure 1-8 shows the layout of a color table atom.

Figure 1-8 The layout of a color table atom



The color table atom contains the following data elements.

Field descriptions

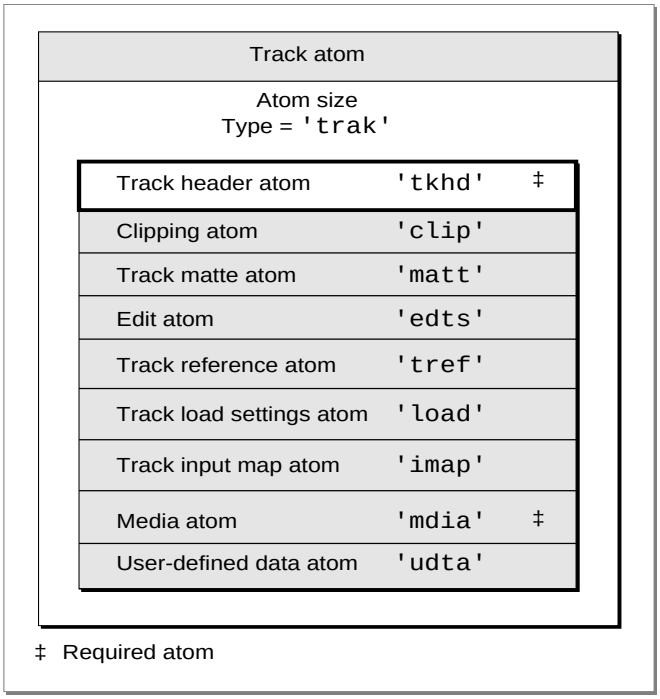
Size	A 32-bit integer that specifies the number of bytes in this color table atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'ctab'`.
Color table seed	A 32-bit integer that must be set to 0.
Color table flags	A 16-bit integer that must be set to 0x8000
Color table size	A 16-bit integer that indicates the number of colors in the following color array. This is a zero-relative value; setting this field to 0 means that there is one color in the array.
Color array	An array of colors. Each color is made up of four unsigned 16-bit integers. The first integer must be set to 0, the second is the red value, the third is the green value, and the fourth is the blue value.

Track Atoms

Track atoms define a single track of a movie. A movie may consist of one or more tracks. Each track is independent of the other tracks in the movie and carries its own temporal and spatial information. Each track atom contains its associated media atom.

Figure 1-9 shows the layout of a track atom. Track atoms have an atom type value of `'trak'`. The track atom requires the track header atom (`'tkhd'`) and the media atom (`'mdia'`). Other child atoms are optional and may include a track clipping atom (`'clip'`), a track matte atom (`'matt'`), an edit atom (`'edts'`), a track reference atom (`'tref'`), a track input map atom (`'imap'`), and a user data atom (`'udta'`).

Figure 1-9 The layout of a track atom



Track atoms contain the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this track atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'trak'`.
Track header	The track header atom associated with this track. See the next section for details.
Track clipping	The track clipping atom associated with this track. See “Clipping Atoms,” beginning on page 19, for more information.
Track matte	The track matte atom associated with this track. See “Track Matte Atoms,” beginning on page 20, for more information.
Edits	The edit atom associated with this track. See “Edit Atoms,” beginning on page 22, for details.
Track references	The track reference atom associated with this track. See “Track Reference Atoms,” beginning on page 26, for details.
Track load settings	The track load settings atom associated with this track. See “Track Load Settings Atoms,” beginning on page 24, for details.
Track input map	The track input map atom associated with this track. See “Track Input Map Atoms,” beginning on page 27, for details.
Media	The media atom associated with this track. See “Media Atoms,” beginning on page 30, for details.

User data — The user data atom associated with this track. See "User Data Atoms" on page 57 for more information.

Track Header Atoms

The track header atom specifies the characteristics of a single track within a movie. A track header atom contains a Size field that specifies the number of bytes and a Type field that indicates the format of the data (defined by the atom type, `'tkhd'`). Figure 1-10 shows the structure of a track header atom.

Figure 1-10 The layout of a track header atom

Track header atom	Bytes
Atom size	4
Type = `'tkhd'`	4
Version	1
Flags	3
Creation time	4
Modification time	4
Track ID	4
Reserved	4
Duration	4
Reserved	8
Layer	2
Alternate group	2
Volume	2
Reserved	2
Matrix structure	36
Track width	4
Track height	4

The track header atom contains the track characteristics for the track, including temporal, spatial, and volume information.

Track header atoms contain the following data elements.

Field descriptions

Size — A 32-bit integer that specifies the number of bytes in this track header atom.

Type — A 32-bit integer that identifies the atom type; this field must be set to `'tkhd'`.

Version — A 1-byte specification of the version of this track header.

Track header flags — Three bytes that are reserved for the track header flags. These flags indicate how the track is used in the movie. The following flags are valid (all flags are enabled when set to 1).

Track enabled
: Indicates that the track is enabled. Flag value is 0x0001.

Track in movie
: Indicates that the track is used in the movie. Flag value is 0x0002.

Track in preview
: Indicates that the track is used in the movie's preview. Flag value is 0x0004.

Track in poster
: Indicates that the track is used in the movie's poster. Flag value is 0x0008.

Creation time — A 32-bit integer that indicates (in seconds since midnight, January 1, 1904) when the track header was created.

Modification time — A 32-bit integer that indicates (in seconds since midnight, January 1, 1904) when the track header was changed.

Track ID — A 32-bit integer that uniquely identifies the track. A value of 0 must never be used for a track ID.

Reserved — A 32-bit integer that is reserved for use by Apple. Set this field to 0.

Duration — A time value that indicates the duration of this track (in the movie's time coordinate system). Note that this property is derived from the track's edits. The value of this field is equal to the sum of the durations of all of the track's edits.

Reserved — An 8-byte value that is reserved for use by Apple. Set this field to 0.

Layer — A 16-bit integer that indicates this track's spatial priority in its movie. The QuickTime Movie Toolbox uses this value to determine how tracks overlay one another. Tracks with lower layer values are displayed in front of tracks with higher layer values.

Alternate group — A 16-bit integer that specifies a collection of movie tracks that contain alternate data for one another. QuickTime chooses one track from the group to be used when the movie is played. The choice may be based on such considerations as playback quality or language and the capabilities of the computer.

Volume — A 16-bit fixed-point value that indicates how loudly this track's sound is to be played. A value of 1.0 indicates normal volume.

Reserved — A 16-bit integer that is reserved for use by Apple. Set this field to 0.

Matrix — The matrix structure associated with this track. See Figure 1-44 on page 77 for an illustration of a matrix structure.

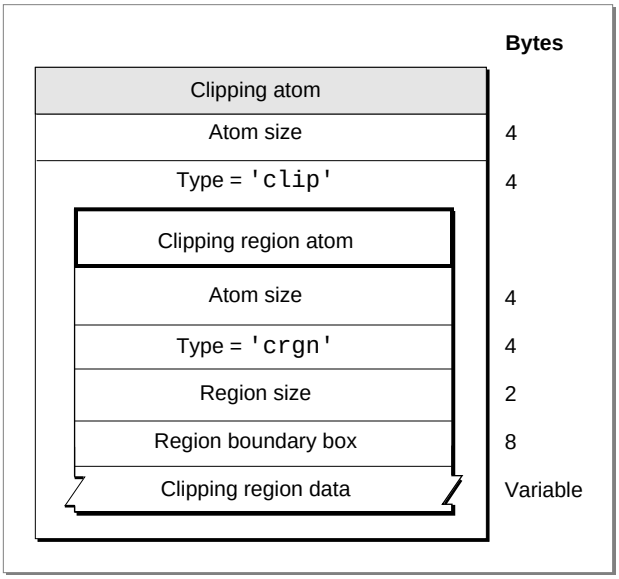
Track width	A 32-bit fixed-point number that specifies the width of this track in pixels.
Track height	A 32-bit fixed-point number that indicates the height of this track in pixels.

Clipping Atoms

Clipping atoms specify the clipping regions for movies and for tracks. The clipping atom has an atom type value of `'clip'`.

Figure 1-11 shows the layout of a clipping atom.

Figure 1-11 The layout of a clipping atom



Clipping atoms contain the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this clipping atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'clip'`.
Clipping region atom	The clipping region atom; these atoms are described in the next section.

Clipping Region Atoms

The clipping region atom contains the data that specifies the clipping region, including its size and bounding box and region. Clipping region atoms have an atom type value of `'crgn'`.

The layout of the clipping region atom is shown in Figure 1-11.

Clipping region atoms contain the following data elements.

Field descriptions

Size — A 32-bit integer that specifies the number of bytes in this clipping region atom.

Type — A 32-bit integer that identifies the atom type; this field must be set to `'crgn'`.

Region size

Region boundary box

Clipping region data

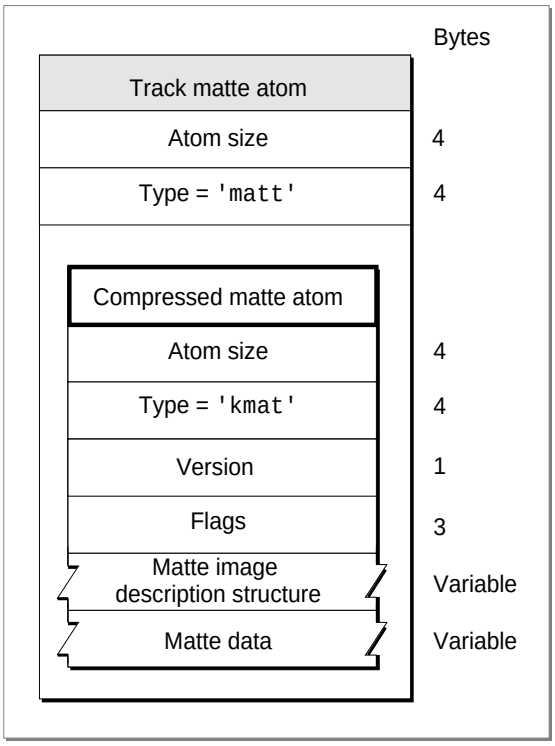
The Region size, Region boundary box, and Clipping region data fields constitute a QuickDraw region. See *Inside Macintosh: Imaging* for details on QuickDraw regions.

Track Matte Atoms

Track matte atoms are used to visually blend the track's image when it is displayed. For a complete description on the use of mattes in QuickTime, see *Inside Macintosh: QuickTime*. Track matte atoms have an atom type value of `'matt'`.

Figure 1-12 shows the layout of a track matte atom.

Figure 1-12 The layout of a track matte atom



Track matte atoms contain the following data elements.

Field descriptions

Size — A 32-bit integer that specifies the number of bytes in this track matte atom.

Type — A 32-bit integer that identifies the atom type; this field must be set to `'matt'`.

Compressed matte atom — The actual matte data in a compressed matte atom. These atoms are described in the next section.

Compressed Matte Atoms

The compressed matte atom specifies the image description structure associated with a particular matte atom. Compressed matte atoms have an atom type value of `'kmat'`.

The layout of the compressed matte atom is shown in Figure 1-12.

Compressed matte atoms contain the following data elements.

Field descriptions

Size — A 32-bit integer that specifies the number of bytes in this compressed matte atom.

Type	A 32-bit integer that identifies the atom type; this field must be set to `'kmat'`.
Version	A 1-byte specification of the version of this compressed matte atom.
Flags	Three bytes of space for flags. Set this field to 0.
Matte image description	An image description structure associated with this matte data. The image description contains detailed information that governs how the matte data is used. See "Video Sample Description" on page 59 for more information about image descriptions.
Matte data	The compressed matte data that is of variable length.

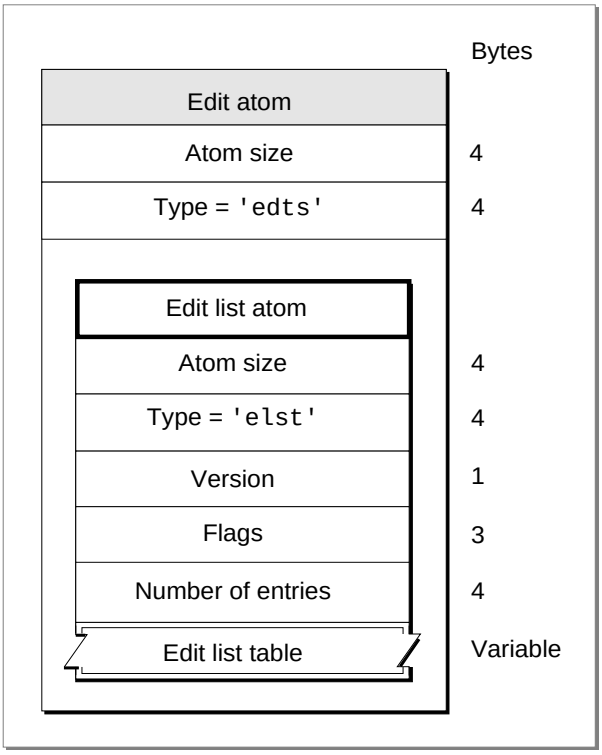
Edit Atoms

You use edit atoms to define the portions of the media that are to be used to build up a track for a movie. The edits themselves are contained in an edit list table, that consists of time offset and duration values for each segment. Edit atoms have an atom type value of `'edts'`.

Figure 1-13 shows the layout of an edit atom.

Note
If the edit atom or the edit list atom is missing, you can assume that the entire media is used by the track. ◆

Figure 1-13 The layout of an edit atom



Edit atoms contain the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this edit atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'edts'`.
Edit list	The edit list atom that contains the edit list information; these atoms are described in the next section.

Edit List Atoms

You use the edit list atom, shown in Figure 1-13, to map from a time in a movie to a time in a media, and ultimately to media data. This information is in the form of an edit list table, shown in Figure 1-14. Edit list atoms have an atom type value of `'elst'`.

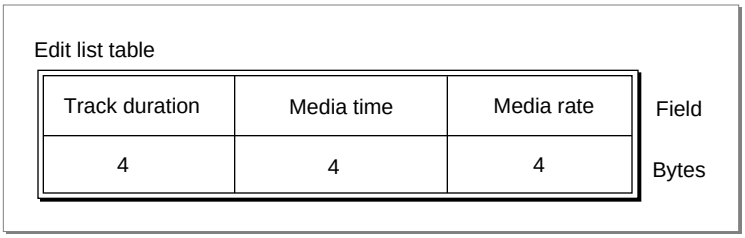
Edit list atoms contain the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this edit list atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'elst'`.

Version	A 1-byte specification of the version of this edit list atom.
Flags	Three bytes of space for flags. Set this field to 0.
Number of entries	A 32-bit integer that specifies the number of entries in the edit list atom that follows.
Edit list table	An array of 32-bit values grouped into entries containing 3 values each.

Figure 1-14 The layout of an edit list table



An edit list table contains the following elements.

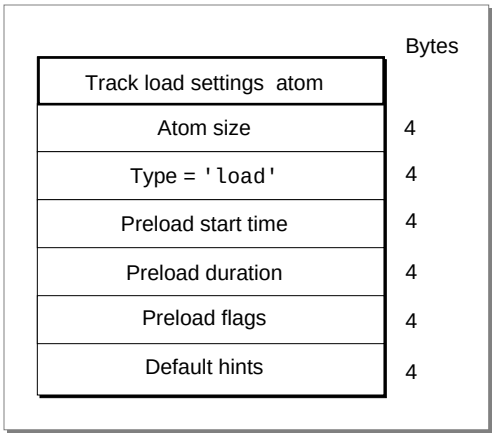
Field descriptions

Track duration	A 32-bit integer that specifies the duration of this edit segment in units of the movie's time scale.
Media time	A 32-bit integer containing the starting time within the media of this edit segment (in media time scale units). If this field is set to –1, it is an empty edit. The last edit in a track should never be an empty edit. Any difference between the movie's duration and the track's duration is expressed as an implicit empty edit.
Media rate	A 32-bit fixed-point number that specifies the relative rate at which to play the media corresponding to this edit segment. This rate value cannot be 0 or negative.

Track Load Settings Atoms

Track load settings atoms contain information that indicates how the track is to be used in its movie. Applications that read QuickTime files can use this information to process the movie data more efficiently. Track load settings atoms are optional and have an atom type value of `'load'`.

Figure 1-15 The layout of a track load settings atom



Track load settings atoms contain the following data elements.

Field descriptions

Size — A 32-bit integer that specifies the number of bytes in this track load settings atom.

Type — A 32-bit integer that identifies the atom type; this field must be set to `'load'`.

Preload start time — A 32-bit integer specifying the starting time, in the movie's time coordinate system, of a segment of the track that is to be preloaded. Used in conjunction with the Preload duration value.

Preload duration — A 32-bit integer specifying the duration, in the movie's time coordinate system, of a segment of the track that is to be preloaded. If the duration is set to –1, it means that the preload segment extends from the preload start time to the end of the track. All media data in the segment of the track defined by the preload start time and preload duration values should be loaded into memory when the movie is to be played.

Preload flags — A 32-bit integer containing flags governing the preload operation. Only two flags are defined and they are mutually exclusive. If Preload flags is set to 1, the track is to be preloaded regardless of whether it is enabled. If Preload flags is set to 2, the track is only preloaded if it is enabled.

Default hints — A 32-bit integer containing playback hints. More than one flag may be enabled. Flags are enabled by setting them to 1. The following flags are defined.

Double buffer — This flag indicates that the track should be played using double-buffered I/O. This flag's value is 0x0020.

High quality — This flag indicates that the track should be displayed at highest possible quality, without regard to real time performance considerations. This flag's value is 0x0100.

Track Reference Atoms

Track reference atoms define relationships between tracks. Track reference atoms allow tracks to specify their relationships to other tracks. For example, if a movie has three video tracks and three sound tracks, track references allow you to identify the related sound and video tracks. Track reference atoms have an atom type value of `'tref'`.

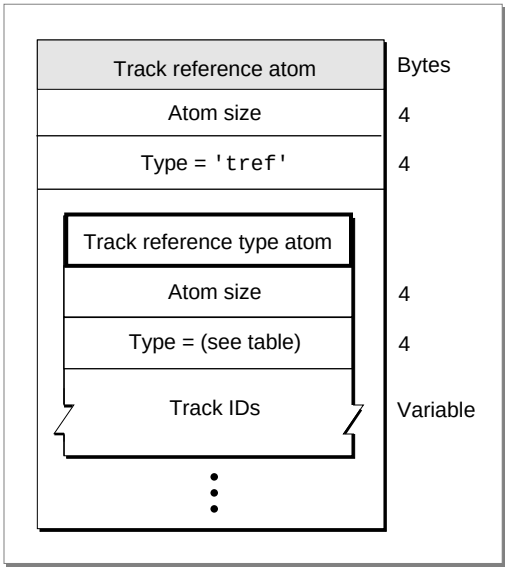
Track references are uni-directional and point from the recipient track to the source track. For example, a video track may reference a time code track to indicate where its time code is stored, but the time code track would not reference the video track. The time code track is the source of time information for the video track.

A single track may reference multiple tracks. For example, a video track could reference a sound track to indicate that the two are synchronized and a time code track to indicate where its time code is stored.

A single track may also be referenced by multiple tracks. For example, both a sound and video track could reference the same time code track if they share the same timing information.

Figure 1-16 shows the layout of a track reference atom.

Figure 1-16 The layout of a track reference atom



A track reference atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this track reference atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'tref'`.

Reference atoms — A list of track reference type atoms containing the track reference information. These atoms are described next.

Each track reference atom defines relationships with tracks of a specific type. The reference type implies a track type. Table 1-2 shows the track reference types and their descriptions.

Table 1-2 Track reference types

Reference type	Description
`'tmcd'`	Time code. Usually references a time code track.
`'chap'`	Chapter or scene list. Usually references a text track.
`'sync'`	Synchronization. Usually between a video and sound track. Indicates that the two tracks are synchronized. The reference can be from either track to the other, or there may be two references.
`'scpt'`	Transcript. Usually references a text track.
`'ssrc'`	Non-primary source. Indicates that the referenced track should send its data to this track, rather than presenting it. The referencing track will use the data to modify how it presents its data. See the next section, "Track Input Map Atoms," for more information.

Each track reference type atom contains the following data elements.

Field descriptions

Size — A 32-bit integer that specifies the number of bytes in this track reference type atom.

Type — A 32-bit integer that identifies the atom type; this field must be set to one of the values shown in Table 1-2.

Track IDs — A list of track ID values specifying the related tracks. Note that this is one case where track ID values may be set to 0. Unused entries in the atom may have a track ID value of 0. Setting the track ID to 0 may be more convenient that deleting the reference.

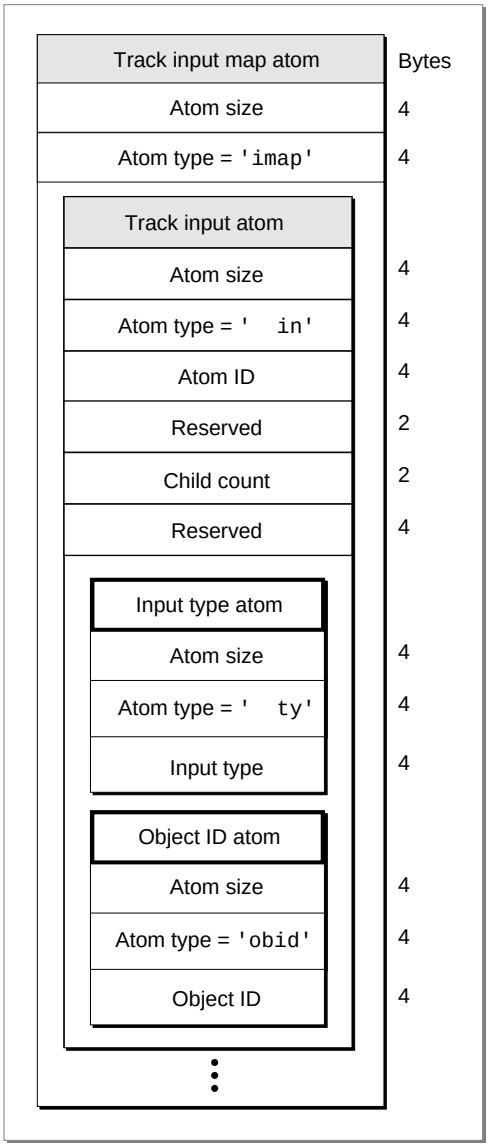
You can determine the number of track references stored in a track reference type atom by subtracting its header size from its overall size and then dividing by the size, in bytes, of a track ID.

Track Input Map Atoms

Track input map atoms define how data being sent to this track from its non-primary sources is to be interpreted. Track references of type `'ssrc'` define a track's secondary data sources. These sources provide additional data that is used when processing the track. Track input map atoms are optional and have an atom type value of `'imap'`.

Figure 1-17 shows the layout of a track input map atom. This atom contains one or more track input atoms. Note that the track input map atom is a QT atom structure.

Figure 1-17 The layout of a track input map atom



Each track input map atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this track input map atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'imap'`.
Track input atoms	A list of track input atoms specifying how to use the input data.

The input map defines all of the track's secondary inputs. Each secondary input is defined using a separate track input atom.

Each track input atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this track input atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `' in'` (note that the two leading bytes must be set to 0x00).
Atom ID	A 32-bit integer relating this track input atom to its secondary input. The value of this field corresponds to the index of the secondary input in the track reference atom. That is, the first secondary input corresponds to the track input atom with an Atom ID value of 1; the second to the track input atom with an Atom ID of 2, and so on.
Reserved	A 16-bit integer that must be set to 0.
Child count	A 16-bit integer specifying the number of child atoms in this atom.
Reserved	A 32-bit integer that must be set to 0.

The track input atom, in turn, may contain two other types of atoms: input type atoms and object ID atoms. The input type atom is required; it specifies how the data is to be interpreted.

The input type atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this input type atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `' ty'` (note that the two leading bytes must be set to 0x00).
Input type	A 32-bit integer that specifies the type of data that is to be received from the secondary data source. Table 1-3 lists valid values for this field.

Table 1-3 Input types

Input identifier	Description
1	A 3x3 transformation matrix to transform the track's location, scaling, and so on.
2	A QuickDraw clipping region to change the track's shape.
3	An 8.8 fixed point value indicating the relative sound volume. This is used for fading the volume.
4	A 16-bit integer indicating the sound balance level. This is used for panning the sound location.
5	A graphics mode record (32-bit integer indicating graphics mode, followed by an RGB color) to modify the track's graphics mode for visual fades.

Table 1-3 Input types (continued)

Input identifier	Description
6	A 3x3 transformation matrix to transform an object within the track's location, scaling, and so on.
7	A graphics mode record (32-bit integer indicating graphics mode, followed by an RGB color) to modify an object within the track's graphics mode for visual fades.
`'vide'`	Compressed image data for an object within the track.

If the input is operating on an object within the track (for example, a sprite within a sprite track), an object ID atom must be included in the track input atom to identify the object.

The object ID atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this object ID atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'obid'`.
Object ID	A 32-bit integer identifying the object.

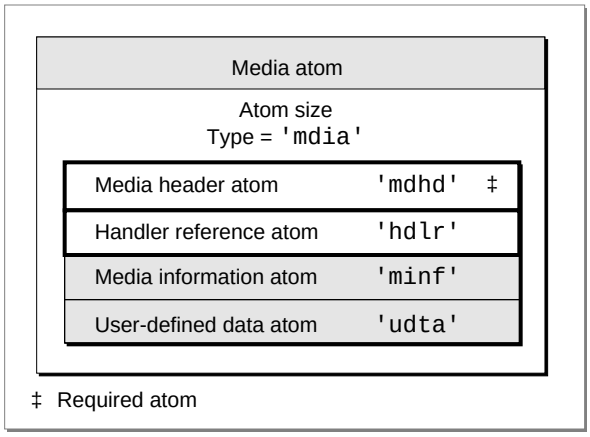
Media Atoms

Media atoms define a track's movie data. The media atom contains information that specifies the media handler component that is to interpret the media data, and it also specifies the data references.

The media atom has an atom type of `'mdia'`. It may contain other atoms, such as a media header (`'mdhd'`), a handler reference (`'hdlr'`), media information (`'minf'`), and user data (`'udta'`). The only required atom in a media atom is the media header atom.

Figure 1-18 shows the layout of a media atom.

Figure 1-18 The layout of a media atom



Note
The handler reference atom tells you the kind of media this media atom contains—for example, video or sound. The layout of the media information atom is specific to the media handler that is to interpret the media. "Media Information Atoms," beginning on page 34, discusses how data may be stored in a media, using the video media format defined by Apple as an example. ◆

Media atoms contain the following data elements.

Field descriptions

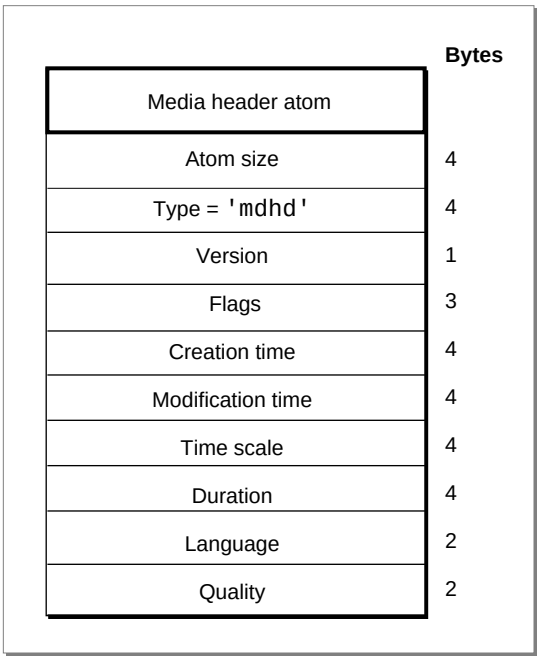
Size	A 32-bit integer that specifies the number of bytes in this media atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'mdia'`.
Media header	The media header atom. This atom contains the standard media information. See the next section for details.
Media handler	The media handler atom. This atom identifies the media handler component that is to be used to interpret the media data. See "Handler Reference Atoms," beginning on page 33, for more information.
Media information	The media information atom. This atom contains media-type specific data for use by the media handler component. See "Media Information Atoms" beginning on page 34 for more information.
User data	The user data atom associated with this media. See "User Data Atoms" on page 57 for more information.

Media Header Atoms

The media header atom specifies the characteristics of a media, including time scale and duration. The media atom has an atom type of `'mdhd'`.

Figure 1-19 shows the layout of a media header atom.

Figure 1-19 The layout of a media header atom



Media header atoms contain the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this media header atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'mdhd'`.
Version	One byte that specifies the version of this movie.
Flags	Three bytes of space for media header flags. Set this field to 0.
Creation time	A 32-bit integer that specifies (in seconds since midnight, January 1, 1904) when the media atom was created.
Modification time	A 32-bit integer that specifies (in seconds since midnight, January 1, 1904) when the media atom was changed.
Time scale	A time value that indicates the time scale for this media—that is, the number of time units that pass per second in its time coordinate system.
Duration	The duration of this media in units of its time scale.
Language	A 16-bit integer that specifies the language code for this media. See “Basic Data Types” on page 75 for valid language codes.

Quality	A 16-bit integer that specifies the media's playback quality—that is, its suitability for playback in a given environment. See *Inside Macintosh: QuickTime* for details on playback quality.

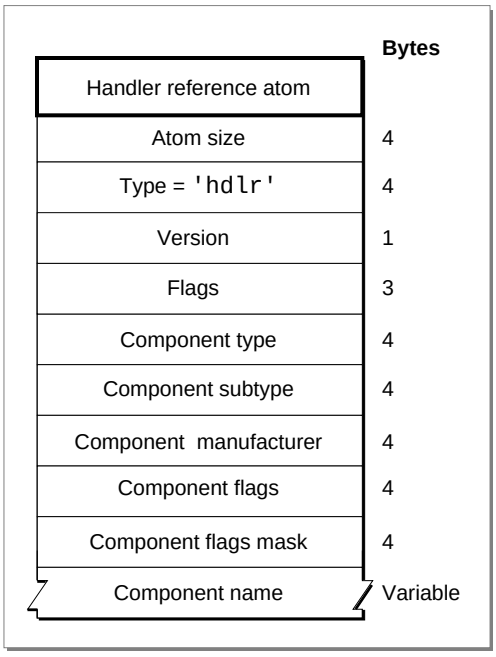
Handler Reference Atoms

The handler reference atom specifies the media handler component that is to be used to interpret the media's data. The handler reference atom has an atom type value of `'hdlr'`.

Historically, the handler reference atom was also used for data references. However, this use may now be ignored.

Figure 1-20 shows the layout of a handler reference atom.

Figure 1-20 The layout of a handler reference atom



Handler reference atoms contain the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this handler reference atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'hdlr'`.
Version	A 1-byte specification of the version of this handler information.
Flags	A 3-byte space for handler information flags. Set this field to 0.

Component type A four-character code that identifies the type of the handler. Only two values are valid for this field: `'mhlr'` for media handlers and `'dhlr'` for data references.

Component subtype
A four-character code that identifies the type of the media handler or data handler. For media handlers, this field defines the type of data, for example, `'vide'` for video data or `'soun'` for sound data.

For data handlers, this field defines the data reference type. For example, a Component subtype value of `'alis'` identifies a file alias.

Component manufacturer
Reserved. Set to 0.

Component flags Reserved. Set to 0.

Component flags mask
Reserved. Set to 0.

Component name A Pascal string that specifies the name of the component—that is, the media handler used when this movie was created. This field may contain a zero-length (empty) string.

Media Information Atoms

Media information atoms (defined by the `'minf'` atom type) store handler-specific information for a track's media data. The media handler uses this information to map from media time to media data and to process the media data.

These atoms contain information that is specific to the type of data defined by the media. Further, the format and content of media information atoms are dictated by the media handler that is responsible for interpreting the media data stream. Another media handler would not know how to interpret this information.

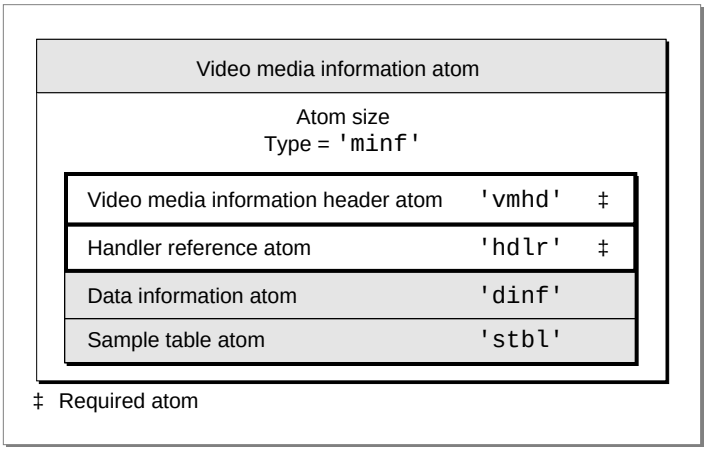
This section describes the atoms that store media information for the video (defined by the `'vmhd'` atom type), sound (defined by the `'smhd'` atom type), and base (defined by the `'gmhd'` atom type) portions of QuickTime movies.

Note
"Using Sample Atoms," beginning on page 56, discusses how the video media handler locates samples in a video media. ◆

Video Media Information Atoms

Video media information atoms are the highest-level atoms in video media. These atoms contain a number of other atoms that define specific characteristics of the video media data. Figure 1-21 shows the layout of a video media information atom.

Figure 1-21 The layout of a media information atom for video



The video media information atom contains the following data elements.

Field descriptions

Size — A 32-bit integer that specifies the number of bytes in this video media information atom.

Type — A 32-bit integer that identifies the atom type; this field must be set to `'minf'`.

Video media information — The video media information header atom (a required atom), which is described in the next section.

Handler reference — The handler reference atom (a required atom), which contains information specifying the data handler component that provides access to the media data. "Handler Reference Atoms" beginning on page 33 discusses handler reference atoms. The handler uses the data information atom to understand the media's data references.

Data information — The data information atom, described in "Data Information Atoms" on page 41.

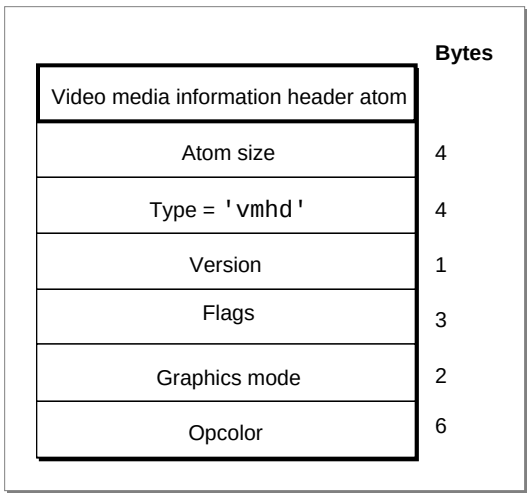
Sample table — The sample table atom, described in "Sample Table Atoms" on page 45.

Video Media Information Header Atoms

Video media information header atoms define specific color and graphics mode information.

Figure 1-22 shows the structure of a video media information header atom.

Figure 1-22 The layout of a media information header atom for video



The video media information header atom contains the following data elements.

Field descriptions

Size — A 32-bit integer that specifies the number of bytes in this video media information header atom.

Type — A 32-bit integer that identifies the atom type; this field must be set to `'vmhd'`.

Version — A 1-byte specification of the version of this video media information header atom.

Flags — A 3-byte space for video media information flags. There is one defined flag.

No lean ahead
: This is a compatibility flag that allows QuickTime to distinguish between movies created for QuickTime 1.0 and newer movies. You should always set this flag to 1, unless you are creating a movie intended for playback using version 1.0 of QuickTime. This flag's value is 0x0001.

Graphics mode — A 16-bit integer that specifies the transfer mode. The transfer mode specifies which Boolean operation QuickDraw should perform when drawing or transferring an image from one location to another.

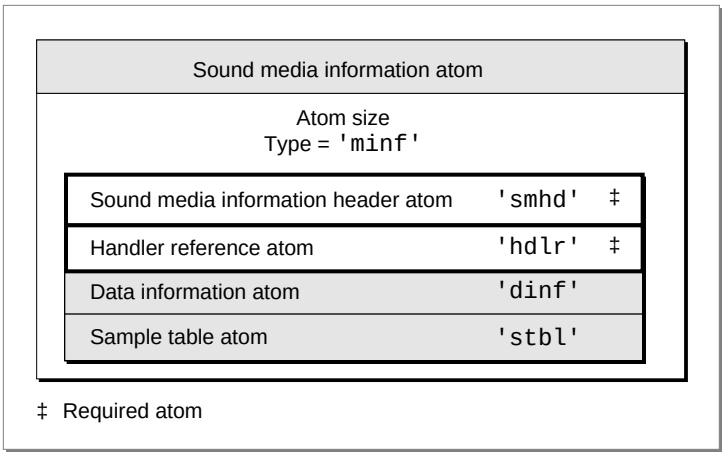
Opcolor — Three 16-bit values that specify the red, green, and blue colors for the transfer mode operation indicated in the Graphics mode field.

For comprehensive details on transfer modes and opcolors and their values in QuickDraw, see *Inside Macintosh: Imaging*.

Sound Media Information Atoms

Sound media information atoms are the highest-level atoms in sound media. These atoms contain a number of other atoms that define specific characteristics of the sound media data. Figure 1-23 shows the layout of a sound media information atom.

Figure 1-23 The layout of a media information atom for sound



The sound media information atom contains the following data elements.

Field descriptions

Size — A 32-bit integer that specifies the number of bytes in this sound media information atom.

Type — A 32-bit integer that identifies the atom type; this field must be set to `'minf'`.

Sound media information — The sound media information header atom (a required atom), which is described in the next section.

Handler reference — The handler reference atom (a required atom), which contains information specifying the data handler component that provides access to the media data. “Handler Reference Atoms” beginning on page 33 discusses handler reference atoms. The handler uses the data information atom to understand the media’s data references.

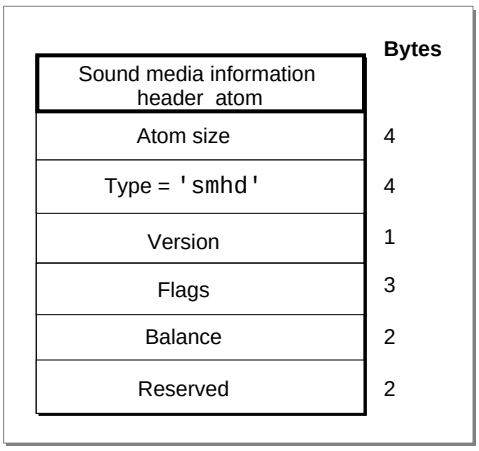
Data information — The data information atom, described in “Data Information Atoms” on page 41.

Sample table — The sample table atom, described in “Sample Table Atoms” on page 45.

Sound Media Information Header Atoms

The sound media information header atom (shown in Figure 1-24) stores the sound media’s control information, such as balance.

Figure 1-24 The layout of a sound media information header atom



The sound media information header atom contains the following data elements.

Field descriptions

Size — A 32-bit integer that specifies the number of bytes in this sound media information header atom.

Type — A 32-bit integer that identifies the atom type; this field must be set to `'smhd'`.

Version — A 1-byte specification of the version of this sound media information header atom.

Flags — A 3-byte space for sound media information flags. Set this field to 0.

Balance — A 16-bit integer that specifies the sound balance of this sound media. Sound balance is the setting that controls the mix of sound between the two speakers of a computer. This field is normally set to 0. See *Inside Macintosh: QuickTime* for more information about QuickTime sound balance values.

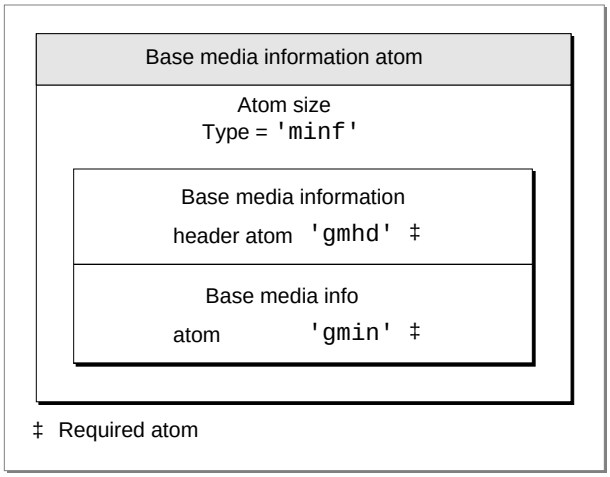
Reserved — Reserved for use by Apple. Set this field to 0.

Base Media Information Atoms

The base media information atom (shown in Figure 1-25) stores the media information for media types, such as text, MPEG, time code, and music.

Media types that are derived from the base media handler may add other atoms within the base media information atom, as appropriate. At present, the only media type that defines any additional atoms is the time code media. See “Time Code Media Information Atom” beginning on page 66 for more information about time code media.

Figure 1-25 The layout of the base media information atom



The base media information atom contains the following data elements.

Field descriptions

Size — A 32-bit integer that specifies the number of bytes in this base media information atom.

Type — A 32-bit integer that identifies the atom type; this field must be set to `'minf'`.

Base media information header — The base media information header atom (a required atom), which is described in the next section.

Base media info — The base media info atom contains control information that describes the media.

Base Media Information Header Atoms

The base media information header atom indicates that this media information atom pertains to a base media.

The base media information header atom contains the following data elements.

Field descriptions

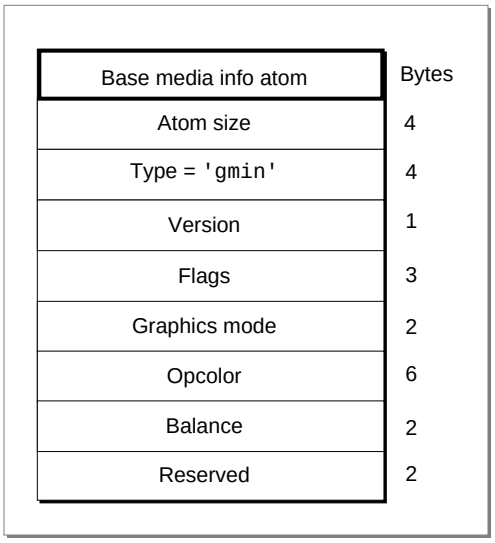
Size — A 32-bit integer that specifies the number of bytes in this base media information header atom.

Type — A 32-bit integer that identifies the atom type; this field must be set to `'gmhd'`.

Base Media Info Atoms

The base media info atom, contained in the base media information atom, defines the media's control information, including graphics mode and balance information. Figure 1-26 shows the layout of a base media info atom.

Figure 1-26 The layout of the base media info atom



The base media info atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this base media information header atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'gmin'`.
Version	A 1-byte specification of the version of this base media information header atom.
Flags	A 3-byte space for base media information flags. Set this field to 0.
Graphics mode	A 16-bit integer that specifies the transfer mode. The transfer mode specifies which Boolean operation QuickDraw should perform when drawing or transferring an image from one location to another.
Opcolor	Three 16-bit values that specify the red, green, and blue colors for the transfer mode operation indicated in the Graphics mode field.
Balance	A 16-bit integer that specifies the sound balance of this media. Sound balance is the setting that controls the mix of sound between the two speakers of a computer. This field is normally set to 0. See *Inside Macintosh: QuickTime* for more information about QuickTime sound balance values.
Reserved	Reserved for use by Apple. Set this field to 0.

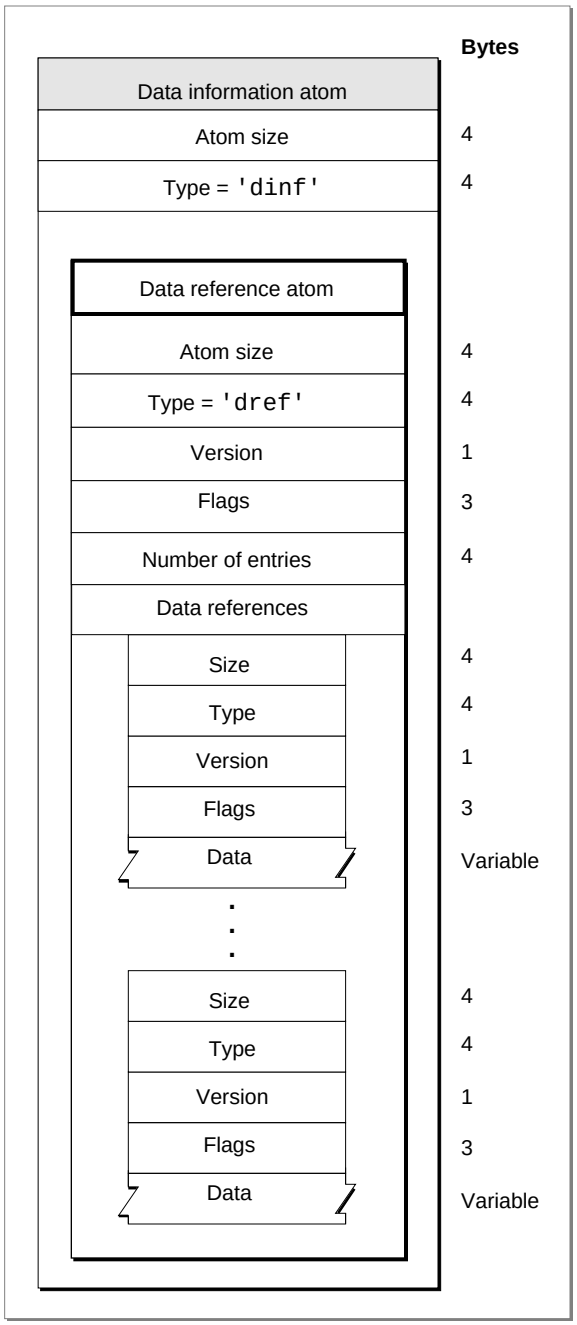
For comprehensive details on transfer modes and opcolors and their values in QuickDraw, see *Inside Macintosh: Imaging*.

Data Information Atoms

The handler reference atom (described in "Handler Reference Atoms," beginning on page 33) contains information specifying the data handler component that provides access to the media data. The data handler component uses the data information atom to interpret the media's data. Data information atoms have an atom type value of `'dinf'`.

Figure 1-27 shows the layout of a data information atom.

Figure 1-27 The layout of a data information atom



The data information atom contains the following data elements.

Field descriptions

Size A 32-bit integer that specifies the number of bytes in this data information atom.

Type	A 32-bit integer that identifies the atom type; this field must be set to `'dinf'`.
Data references	A data reference atom, described in the next section, contains the data references.

Data Reference Atoms

Data reference atoms contain tabular data that instructs the data handler component how to access the media's data. Figure 1-27 shows the data reference atom.

The data reference atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this data reference atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'dref'`.
Version	A 1-byte specification of the version of this data reference atom.
Flags	A 3-byte space for data reference flags. Set this field to 0.
Number of entries	A 32-bit integer containing the count of data references that follow.
Data references	An array of data references.

Each data reference is formatted like an atom and contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in these data references.
Type	A 32-bit integer that specifies the type of the data in the data references. Table 1-4 lists valid Type values.
Version	A 1-byte specification of the version of these data references.
Flags	A 3-byte space for data reference flags. There is one defined flag.
Self reference	This flag indicates that the media's data is in the same file as the movie atom. On the Macintosh, and other file systems with multifork files, set this flag to 1 even if the data resides in a different fork from the movie atom. This flag's value is 0x0001.
Data references	The data reference information.

Table 1-4 shows the currently defined data reference types that may be stored in a movie.

Table 1-4 Data reference types

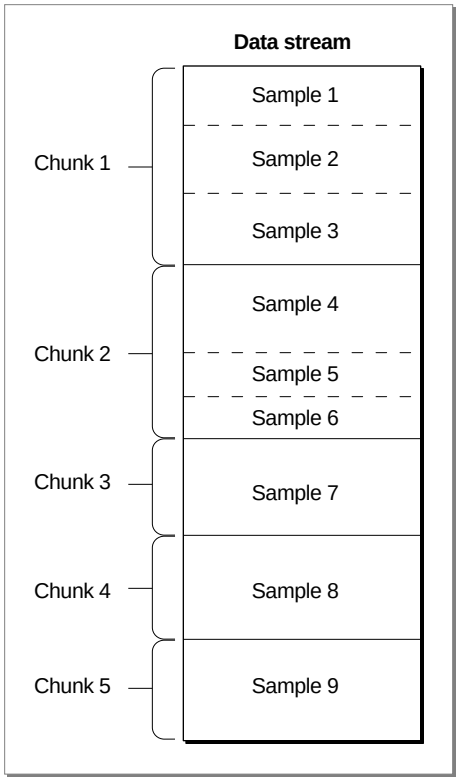
Data reference type	Description
`'alis'`	Data reference is a Macintosh alias. An alias contains information about the file, including its full path name. For more information, see *Inside Macintosh: Files*.
`'rsrc'`	Data reference is a Macintosh alias. Appended to the end of the alias is the resource type (stored as a 32-bit unsigned integer) and ID (stored as a 16-bit signed integer) to use within the specified file.

Sample Atoms

QuickTime stores media data in samples. A sample is a single element in a sequence of time-ordered data. Samples are stored in the media, and they may have varying durations.

Figure 1-28 shows the way that samples are stored in a series of **chunks** in a media.

Figure 1-28 Samples in a media



Chunks are a collection of data samples in a media that allow optimized data access. A chunk may contain one or more samples. Chunks in a media may have different sizes, and the individual samples within a chunk may have different sizes from one another.

One way to describe a sample is to use a sample table atom. The sample table atom acts as a storehouse of information about the samples and contains a number of different types of atoms. The various atoms contain information that allows the media handler to parse the samples in the proper order. This approach enforces an ordering of the samples without requiring that the sample data be stored sequentially with respect to movie time in the actual data stream.

The next section discusses the sample table atom. Subsequent sections discuss each of the atoms that may reside in a sample table atom.

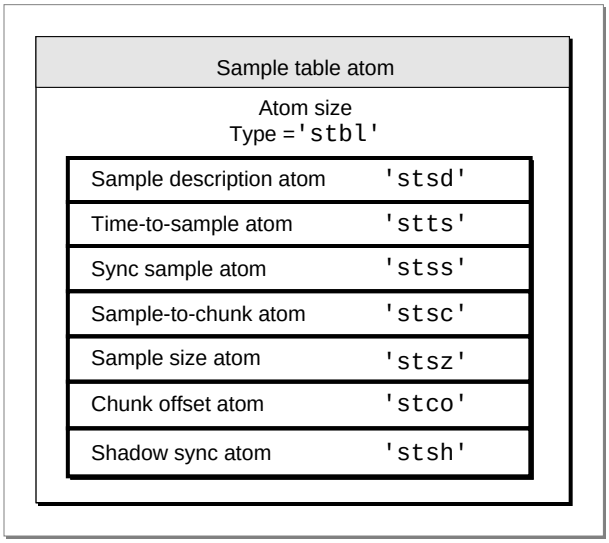
Sample Table Atoms

The sample table atom contains information for converting from media time to sample number to sample location. This atom also indicates how to interpret the sample (for example, whether to decompress the video data and, if so, how). This section describes the format and content of the sample table atom.

The sample table has an atom type of `'stbl'`. It contains the sample description atom, the time-to-sample atom, the sample-to-chunk atom, the sync sample atom, the sample size atom, the chunk offset atom, and the shadow sync atom.

Figure 1-29 shows the layout of a sample table atom.

Figure 1-29 The layout of a sample table atom



The sample table atom contains the following data elements.

Field descriptions

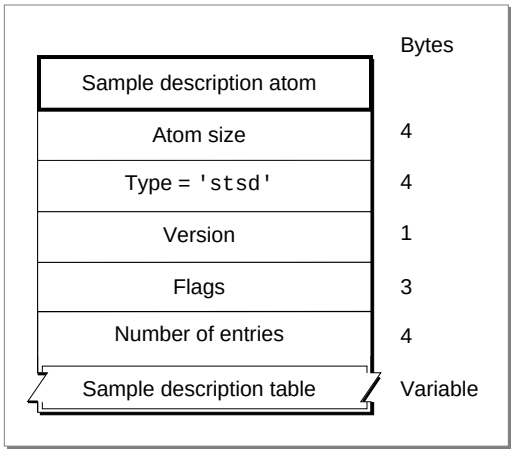
Size	A 32-bit integer that specifies the number of bytes in this sample table atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'stbl'`.
Sample description	The sample description atom, described in the next section.
Time-to-sample	The time-to-sample atom, described in "Time-to-Sample Atoms," beginning on page 48.
Sync sample	The sync sample atom, described in "Sync Sample Atoms," beginning on page 50.
Sample-to-chunk	The sample-to-chunk atom, described in "Sample-to-Chunk Atoms," beginning on page 51.
Sample size	The sample size atom, described in "Sample Size Atoms," beginning on page 53.
Chunk offset	A chunk offset atom, described in "Chunk Offset Atoms," beginning on page 55.
Shadow sync	The shadow sync atom. This atom is obsolete.

Sample Description Atoms

The sample description atom stores information that allows you to decode samples in the media. The data stored in the sample description varies, depending on the media type. For example, in the case of video media, the sample descriptions are image description structures. The sample description information for each media type is explained later in this document, in “Media Data Atom Types” beginning on page 59.

Figure 1-30 shows the layout of a sample description atom.

Figure 1-30 The layout of a sample description atom



The sample description atom has an atom type of `'stsd'`. The sample description atom contains a table of sample descriptions. A media may have one or more sample descriptions, depending upon the number of different encoding schemes used in the media and on the number of files used to store the data. The sample-to-chunk atom identifies the sample description for each sample in the media by specifying the index into this table for the appropriate description (see “Sample-to-Chunk Atoms,” beginning on page 51).

The sample description atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this sample description atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'stsd'`.
Version	A 1-byte specification of the version of this sample description atom.
Flags	A 3-byte space for sample description flags. Set this field to 0.
Number of entries	A 32-bit integer containing the number of sample descriptions that follow.

Sample description table
: An array of sample descriptions.

While the exact format of the sample description varies by media type, the first four fields of every sample description table are the same.

Field descriptions

Sample description size
: A 32-bit integer indicating the number of bytes in the sample description.

Data format
: A 32-bit integer indicating the format of the stored data. This depends on the media type, but is usually either the compression format or the media type.

Reserved
: Six bytes that must be set to 0.

Data reference index
: A 16-bit integer that contains the index of the data reference to use to retrieve data associated with samples that use this sample description. Data references are stored in data reference atoms.

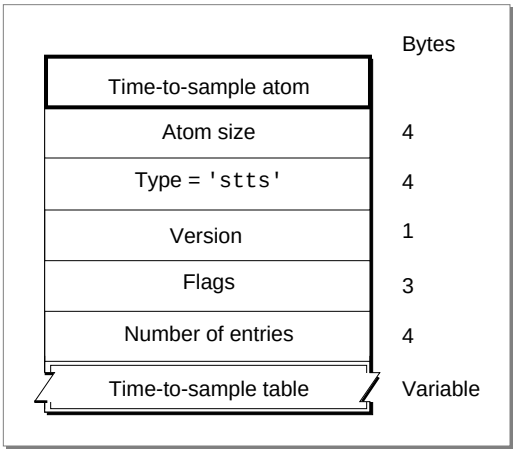
Time-to-Sample Atoms

Time-to-sample atoms store duration information for a media's samples, providing a mapping from a time in the media to the corresponding data sample. The time-to-sample atom has an atom type of `'stts'`.

You can determine the appropriate sample for any time in a media by examining the time-to-sample atom table, which is contained in a time-to-sample atom.

Figure 1-31 shows the layout of a time-to-sample atom.

Figure 1-31 The layout of a time-to-sample atom

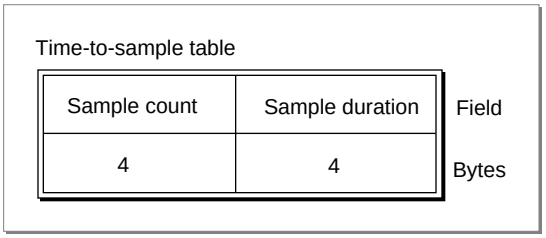


The time-to-sample atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this time-to-sample atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'stts'`.
Version	A 1-byte specification of the version of this time-to-sample atom.
Flags	A 3-byte space for time-to-sample flags. Set this field to 0.
Number of entries	A 32-bit integer containing the number of entries in the time-to-sample table.
Time-to-sample table	A table that defines the duration of each sample in the media. Each table entry contains a count field and a duration field. The structure of a time-to-sample table is shown in Figure 1-32.

Figure 1-32 The layout of a time-to-sample table



You define a time-to-sample table by specifying these entries:

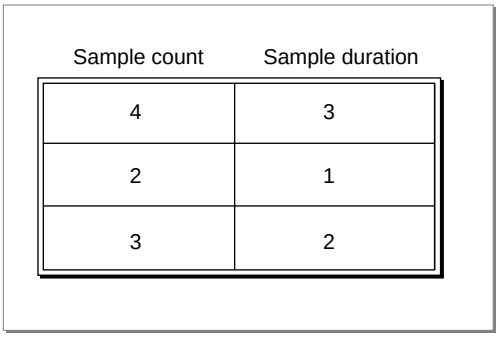
Field descriptions

Sample count	A 32-bit integer that specifies the number of consecutive samples that have the same duration.
Sample duration	A 32-bit integer that specifies the duration of each sample.

Entries in the table describe samples according to their order in the media and their duration. If consecutive samples have the same duration, a single table entry may be used to define more than one sample. In these cases, the count field indicates the number of consecutive samples that have the same duration. For example, if a video media has a constant frame rate, this table would have one entry and the count would be equal to the number of samples.

Figure 1-33 presents an example of a time-to-sample table that is based on the chunked media data shown in Figure 1-28 on page 45. That data stream contains a total of nine samples that correspond in count and duration to the entries in the table shown here. Even though samples 4, 5, and 6 are in the same chunk, sample 4 has a duration of 3, and samples 5 and 6 have a duration of 2.

Figure 1-33 An example of a time-to-sample table



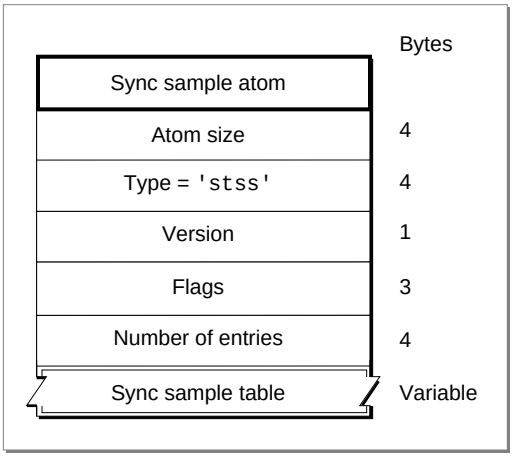
Sync Sample Atoms

The sync sample atom identifies the **key frames** in the media. In a media that contains compressed data, key frames define starting points for portions of a temporally compressed sequence. The key frame is self-contained—that is, it is independent of preceding frames. Subsequent frames may depend on the key frame.

Sync sample atoms have an atom type of `'stss'`. The sync sample atom contains a table of sample numbers. Each entry in the table identifies a sample that is a key frame for the media. Figure 1-34 shows the layout of a sync sample atom.

If no sync sample atom exists, then all the samples are key frames.

Figure 1-34 The layout of a sync sample atom

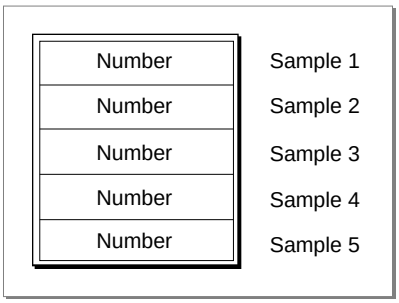


The sync sample atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this sync sample atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'stss'`.
Version	A 1-byte specification of the version of this sync sample atom.
Flags	A 3-byte space for sync sample flags. Set this field to 0.
Number of entries	A 32-bit integer containing the number of entries in the sync sample table.
Sync sample table	A table of sample numbers; each sample number corresponds to a key frame. Figure 1-35 shows the layout of a sync sample table.

Figure 1-35 The layout of a sync sample table

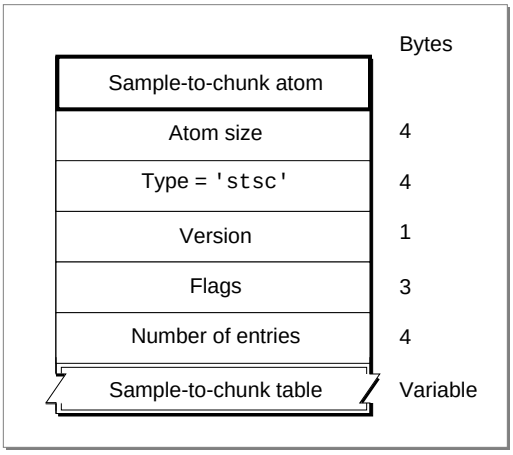


Sample-to-Chunk Atoms

As samples are added to a media, they are collected into chunks that allow optimized data access. A chunk may contain one or more samples. Chunks in a media may have different sizes, and the samples within a chunk may have different sizes. The sample-to-chunk atom stores chunk information for the samples in a media.

Sample-to-chunk atoms have an atom type of `'stsc'`. The sample-to-chunk atom contains a table that maps samples to chunks in the media data stream. Figure 1-36 shows the layout of a sample-to-chunk atom. By examining the sample-to-chunk atom, you can determine the chunk that contains a specific sample.

Figure 1-36 The layout of a sample-to-chunk atom



The sample-to-chunk atom contains the following data elements.

Field descriptions

Size — A 32-bit integer that specifies the number of bytes in this sample-to-chunk atom.

Type — A 32-bit integer that identifies the atom type; this field must be set to `'stsc'`.

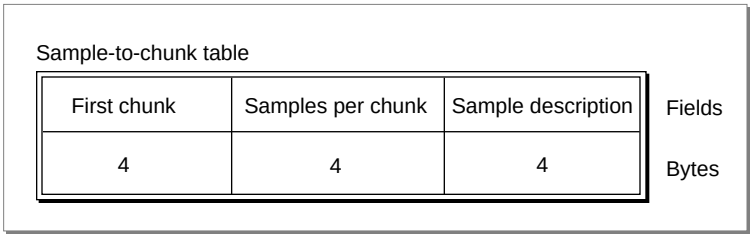
Version — A 1-byte specification of the version of this sample-to-chunk atom.

Flags — A 3-byte space for sample-to-chunk flags. Set this field to 0.

Number of entries — A 32-bit integer containing the number of entries in the sample-to-chunk table.

Sample-to-chunk table — A table that maps samples to chunks. Figure 1-37 shows the structure of a sample-to-chunk table. Each sample-to-chunk atom contains such a table, which identifies the chunk for each sample in a media. Each entry in the table contains a first chunk field, a samples per chunk field, and a sample description ID field. From this information, you can ascertain where samples reside in the media data.

Figure 1-37 The layout of a sample-to-chunk table



You define a sample-to-chunk table by specifying the following data elements.

Field descriptions

First chunk — The first chunk number using this table entry.

Samples per chunk — The number of samples in each chunk.

Sample description ID — The identification number associated with the sample description for the sample. For details on sample description atoms, see "Sample Description Atoms," beginning on page 47.

Figure 1-38 shows an example of a sample-to-chunk table that is based on the data stream shown in Figure 1-28 on page 45.

Figure 1-38 An example of a sample-to-chunk table

Sample-to-chunk table

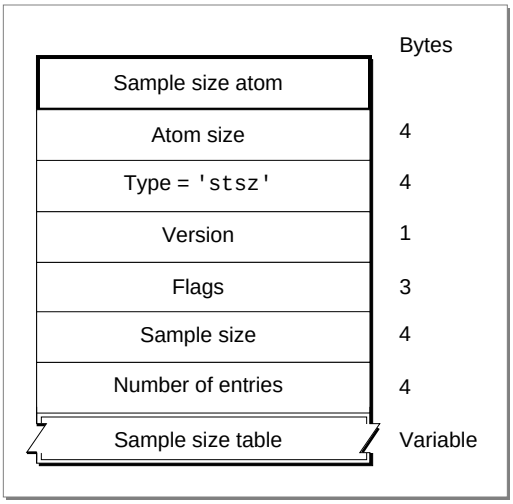
First chunk	Samples per chunk	Sample description ID
1	3	23
3	1	23
5	1	24

Each table entry corresponds to a set of consecutive chunks, each of which contains the same number of samples. Furthermore, each of the samples in these chunks must use the same sample description. Whenever the number of samples per chunk or the sample description changes, you must create a new table entry. If all the chunks have the same number of samples per chunk and use the same sample description, this table has one entry.

Sample Size Atoms

You use sample size atoms to specify the size of each sample in the media. Sample size atoms have an atom type of `'stsz'`. Figure 1-39 shows the layout of a sample size atom.

Figure 1-39 The layout of a sample size atom



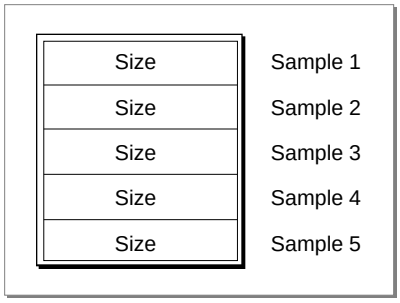
The sample size atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this sample size atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'stsz'`.
Version	A 1-byte specification of the version of this sample size atom.
Flags	A 3-byte space for sample size flags. Set this field to 0.
Sample size	A 32-bit integer specifying the sample size. If all the samples are the same size, this field contains that size value. If this field is set to 0, then the samples have different sizes, and those sizes are stored in the sample size table.
Number of entries	A 32-bit integer containing the number of entries in the sample size table.
Sample size table	A table containing the sample size information. The sample size table contains an entry for every sample in the media's data stream. Each table entry contains a size field. The size field contains the size, in bytes, of the sample in question. The table is indexed by sample number—the first entry corresponds to the first sample, the second entry is for the second sample, and so on.

Figure 1-40 shows a sample size table.

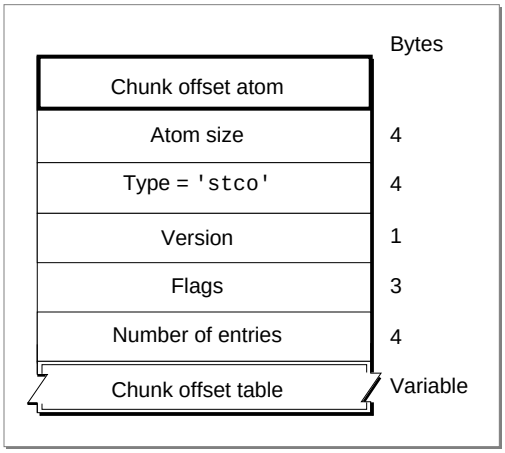
Figure 1-40 An example of a sample size table



Chunk Offset Atoms

Chunk offset atoms identify the location of each chunk of data in the media's data stream. Chunk offset atoms have an atom type of `'stco'`. The chunk offset atom (shown in Figure 1-41) contains a table of offset information.

Figure 1-41 The layout of a chunk offset atom



The chunk offset atom contains the following data elements.

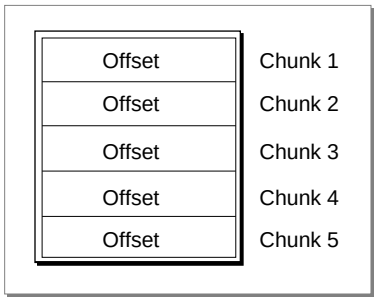
Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this chunk offset atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'stco'`.
Version	A 1-byte specification of the version of this chunk offset atom.
Flags	A 3-byte space for chunk offset flags. Set this field to 0.
Number of entries	A 32-bit integer containing the number of entries in the chunk offset table.

Chunk offset table — A chunk offset table consisting of an array of offset values. There is one table entry for each chunk in the media. The offset contains the byte offset from the beginning of the data stream to the chunk. The table is indexed by chunk number—the first table entry corresponds to the first chunk, the second table entry is for the second chunk, and so on.

Figure 1-42 shows an example of a chunk offset table

Figure 1-42 An example of a chunk offset table



Using Sample Atoms

This section presents examples using the atoms just described. These examples are intended to help you understand the relationships between these atoms. The first section, "Finding a Sample," describes the steps that the video media handler uses to find the sample that contains the media data for a particular time in a media. The second section, "Finding a Key Frame," describes the steps that the video media handler uses to find an appropriate key frame for a specific time in a movie.

Finding a Sample

When QuickTime displays a movie or track, it tells the appropriate media handler to access the media data for a particular time. The media handler must correctly interpret the data stream to retrieve the requested data. In the case of video media, the media handler traverses several atoms to find the location and size of a sample for a given media time. The media handler does the following:

1. Determines the time in the media time coordinate system.
2. Examines the time-to-sample atom to determine the sample number that contains the data for the specified time.
3. Scans the sample-to-chunk atom to discover which chunk contains the sample in question.
4. Extracts the offset to the chunk from the chunk offset atom.
5. Finds the offset within the chunk and the sample's size by using the sample size atom.

Finding a Key Frame

Finding a key frame for a specified time in a movie is slightly more complicated than finding a sample for a specified time. The media handler must use the sync sample atom and the time-to-sample atom together in order to find a key frame. The media handler does the following:

1. Examines the time-to-sample atom to determine the sample number that contains the data for the specified time.
2. Scans the sync sample atom to find the key frame that precedes the sample number chosen in step 1.
3. Scans the sample-to-chunk atom to discover which chunk contains the key frame.
4. Extracts the offset to the chunk from the chunk offset atom.
5. Finds the offset within the chunk and the sample's size by using the sample size atom.

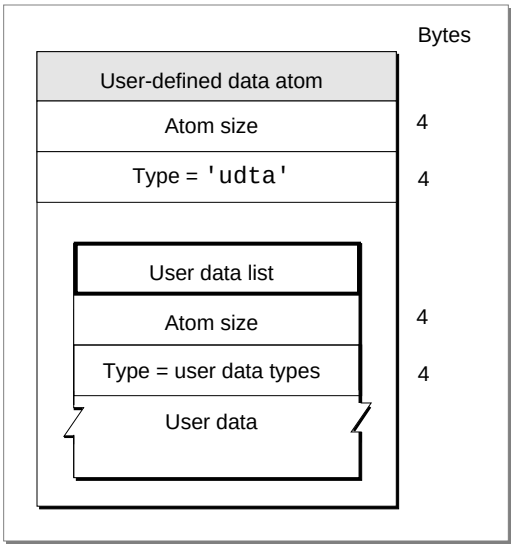
User Data Atoms

User data atoms allow you to define and store arbitrary data associated with a QuickTime object, such as a movie, track, or media. The user data atom has an atom type of `'udta'`.

Inside the user data atom is a list of atoms describing each piece of user data. User data provides a simple way to extend what is stored in a QuickTime movie. For example, you may use user data atoms to store a movie's window position, playback characteristics, or creation information.

Figure 1-43 shows the layout of a user data atom.

Figure 1-43 The layout of a user data atom



The user data atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this user data atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'udta'`.
User data list	A user data list that is itself formatted like a series of atoms. Each data element in the private data portion of the user-defined data atom contains size and type information along with the data. Furthermore, for historical reasons the list of atoms is optionally terminated by a 32-bit integer set to 0. If you are writing a program to read user data atoms, you should allow for the terminating 0. However, if you are writing a program to create user data atoms, you can safely leave out the trailing 0. Table 1-5 lists the currently defined list entry types.

Table 1-5 User data list entry types

List entry type	Description
`'©cpy'`	Copyright statement.
`'©day'`	Date the movie content was created.
`'©dir'`	Name of movie's director.
`'©ed1'` to `'©ed9'`	Edit dates and descriptions.
`'©fmt'`	Indication of movie format (computer-generated, digitized, and so on).
`'©inf'`	Information about the movie.
`'©prd'`	Name of movie's producer.
`'©prf'`	Names of performers.
`'©req'`	Special hardware and software requirements.
`'©src'`	Credits for those who provided movie source content.
`'©wrt'`	Name of movie's writer
`'WLOC'`	Default window location for movie. Two 16 bit values, {x,y}.
`'name'`	Name of object.
`'LOOP'`	Long integer indicating looping style. 0 for none, 1 for looping, 2 for palindrome looping.
`'SelO'`	Play selection only. Byte indicating that only the selected area of the movie be played.
`'AllF'`	Play all frames. Byte indicating that all frames of video should be played, regardless of timing.

All user data list entries whose type begins with the '©' character (ASCII 169), are defined to be international text. These list entries must contain a list of text strings with associated language codes. By storing multiple versions of the same text, a single user data text item can contain translations for different languages.

The list of text strings uses a small integer atom format, which is identical to the QuickTime atom format, except that it uses 16-bit values for size and type instead of 32-bit values. The first value is the size of the string, including the size and type, and the second value is the language code for the string.

Media Data Atom Types

QuickTime uses atoms of different types to store different types of media data—video media for video data, sound media for audio data, and so on. The following sections discuss each of these different media data atom types.

Video Media

Video media is used to store compressed and uncompressed image data in QuickTime movies. It has a media type of `'vide'`.

Video Sample Description

The video sample description contains information that defines how to interpret video media data. This sample description is based on the standard sample description header, as described in “Sample Table Atoms” beginning on page 45.

The Data format field indicates the type of compression that was used to compress the image data. Table 1-6 shows some of the formats currently supported.

Table 1-6 Image compression formats

Compression type	Description
`'cvid'`	Cinepak
`'jpeg'`	JPEG
`'raw '`	Uncompressed RGB
`'YUV2'`	Uncompressed YUV422
`'smc '`	Graphics
`'rle '`	Animation
`'rpza'`	Apple Video
`'kpcd'`	Kodak Photo CD
`'qdgx'`	QuickDraw GX

Table 1-6 Image compression formats (continued)

Compression type	Description
`'mpeg'`	MPEG still image
`'mjpa'`	Motion-JPEG (Format A)
`'mjpb'`	Motion-JPEG (Format B)

The video media handler also adds some of its own fields to the sample description. For more information about each of these fields, see the discussion of image descriptions in *Inside Macintosh: QuickTime*.

Field descriptions

Version	A 16-bit integer indicating the version number of the compressed data. Usually this is set to 0, unless a compressor has changed its data format.
Revision level	A 16-bit integer that must be set to 0.
Vendor	A 32-bit integer that specifies the developer of the compressor that generated the compressed data. Often this field contains `'appl'` to indicate Apple Computer, Inc.
Temporal quality	A 32-bit integer containing a value from 0 to 1023 indicating the degree of temporal compression.
Spatial quality	A 32-bit integer containing a value from 0 to 1023 indicating the degree of spatial compression.
Width	A 16-bit integer that specifies the width of the source image in pixels.
Height	A 16-bit integer that specifies the height of the source image in pixels.
Horizontal resolution	A 32-bit fixed point number containing the horizontal resolution of the image in pixels per inch.
Vertical resolution	A 32-bit fixed point number containing the vertical resolution of the image in pixels per inch.
Data size	A 32-bit integer that must be set to 0.
Frame count	A 16-bit integer that indicates how many frames of compressed data are stored in each sample. Usually set to 1.
Compressor name	A 32-byte Pascal string containing the name of compressor that created the image, such as `"jpeg"`.
Depth	A 16-bit integer that indicates the pixel depth of the compressed image. Values of 1, 2, 4, 8,16, 24, and 32 indicate the depth of color images. The value of 32 should be used only if the image contains an alpha channel. Values of 34, 36, and 40 indicate 2-, 4-, and 8-bit grayscale, respectively, for grayscale images.
Color table ID	A 16-bit integer that identifies which color table to use. If this field is set to –1, the default color table should be used for the specified depth. For all depths below 16 bits per pixel, this indicates a

standard Macintosh color table for the specified depth. Depths of 16, 24, and 32 have no color table.

If the color table ID is set to 0, a color table is contained within the sample description itself. The color table immediately follows the Color table ID field in the sample description. See "Color Table Atoms" on page 14 for a complete description of a color table.

Video sample descriptions can be extended by appending other atoms. These atoms are placed after the color table, if one is present. These extensions to the sample description may contain display hints for the decompressor or may simply carry additional information associated with the images. Table 1-7 lists the currently defined extensions to video sample descriptions.

Table 1-7 Video sample description extensions

Extension type	Description
`'gama'`	A 32-bit fixed point number indicating the gamma level at which the image was captured. The decompressor can use this value to gamma-correct at display time.
`'fiel'`	A 2-byte value indicating the number of video fields in the data stream, and how those fields are to be used. The first byte specifies the field count, and may be set to 1 or 2. The second byte defines the field dominance, as follows: 0–Field dominance unknown 1–Top field is first, temporally 2–Bottom field is first, temporally This information is used by applications that may be modifying decompressed image data, or by decompressor components to determine field display order.
`'mjqt'`	The default quantization table for a Motion JPEG data stream.
`'mjht'`	The default Huffman table for a Motion JPEG data stream.

Video Sample Data

The format of the data stored in video samples is completely dependent on the type of the compressed data stored in the video sample description. The following sections discuss each of the video encoding schemes supported by QuickTime.

Uncompressed RGB

Uncompressed RGB data is stored in a variety of different formats. The format used depends on the Depth field of the video sample description. For all depths, the image data is padded on each scan line to ensure that each scan line begins on an even byte boundary.

- For depths of 1, 2, 4, and 8, the values stored are indexes into the color table specified in the Color table id field.
- For a depth of 16, the pixels are stored as 5-5-5 RGB values with the high bit of each 16-bit integer set to 0.
- For a depth of 24, the pixels are stored packed together in RGB order.
- For a depth of 32, the pixels are stored with an 8-bit alpha channel, followed by 8-bit RGB components.

Uncompressed YUV

Uncompressed YUV data is stored as YUV422 data. Two pixels are combined into a single 32-bit integer stored in YUYV order.

JPEG

QuickTime stores JPEG images according to the rules described in the ISO JPEG specification, document number DIS 10918-1.

Motion JPEG

Motion JPEG (M-JPEG) is a variant of the ISO JPEG specification for use with digital video streams. Instead of compressing an entire image into a single bitstream, M-JPEG compresses each video field separately, returning the resulting JPEG bitstreams consecutively in a single frame.

There are two flavors of M-JPEG currently in use. These two formats differ based on their use of markers. **M-JPEG format A** supports markers; **M-JPEG format B** does not. The following paragraphs describe how QuickTime stores M-JPEG sample data.

Each field of M-JPEG Format A fully complies with the ISO JPEG specification, and therefore supports application markers. QuickTime uses the app 1 marker to store control information, as follows (all of the fields are 32-bit integers):

Field descriptions

Reserved	Contents unpredictable; should be set to 0.
Tag	Identifies the data type; this field must be set to `'mjpg'`.
Field size	Contains the actual size of the image data for this field, in bytes.
Padded field size	Contains the size of the image data, including pad bytes. Some video hardware may append pad bytes to the image data; this field, along with the Field size field, allows you to compute how many pad bytes were added.
Offset to next field	Specifies the offset, in bytes, from the start of the field data to the start of the next field in the bitstream. This field should be set to 0 in the second field's marker data.
Quantization table offset	Specifies the offset, in bytes, from the start of the field data to the quantization table marker. If this field is set to 0, check the image description for a default quantization table (see "Video Sample Description" beginning on page 59 for details).

Huffman table offset
: Specifies the offset, in bytes, from the start of the field data to the Huffman table marker. If this field is set to 0, check the image description for a default Huffman table (see "Video Sample Description" beginning on page 59 for details).

Start of image offset
: Specifies the offset from the start of the field data to the start of image marker. This field should never be set to 0.

M-JPEG Format B does not support markers. In place of the marker, therefore, QuickTime inserts a header at the beginning of the bitstream. Again, all of the fields are 32-bit integers.

Field descriptions

Reserved	Contents unpredictable; should be set to 0.
Tag	Identifies the data type; this field must be set to `'mjpg'`.
Field size	Contains the actual size of the image data for this field, in bytes.
Padded field size	Contains the size of the image data, including pad bytes. Some video hardware may append pad bytes to the image data; this field, along with the Field size field, allows you to compute how many pad bytes were added.
Offset to next field	Specifies the offset, in bytes, from the start of the field data to the start of the next field in the bitstream. This field should be set to 0 in the second field's header data.
Quantization table offset	Specifies the offset, in bytes, from the start of the field data to the quantization table. If this field is set to 0, check the image description for a default quantization table (see "Video Sample Description" beginning on page 59 for details).
Huffman table offset	Specifies the offset, in bytes, from the start of the field data to the Huffman table. If this field is set to 0, check the image description for a default Huffman table (see "Video Sample Description" beginning on page 59 for details).
Start of image offset	Specifies the offset from the start of the field data to the field's image data. This field should never be set to 0.
Reserved	Must be set to 0.
Reserved	Must be set to 0.

The M-JPEG Format B header must be a multiple of 16 in size. When you add pad bytes to the header, set them to 0.

Note

Because this format does not support markers, there is no need to stuff the bitstream with null bytes (0x00) after data bytes that are set to 0xFF. ◆

MPEG Still Images

QuickTime stores MPEG still images according to the Blue Book specification for Enhanced CDs.

Sound Media

Sound media is used to store compressed and uncompressed audio data in QuickTime movies. It has a media type of `'soun'`.

Sound Sample Description

The sound sample description contains information that defines how to interpret sound media data. This sample description is based on the standard sample description header, as described in "Sample Table Atoms" beginning on page 45.

The data format field contains the format of the audio data. Table 1-8 shows some of the current formats.

Table 1-8 Sound data format types

Data format	Description
`'raw '`	Samples are stored uncompressed in offset-binary format (values range from 0 to 255; 128 is silence).
`'twos'`	Samples are stored uncompressed, in twos-complement format (sample values range from –128 to 127 for 8-bit audio, and -32768 to 32767 for 1-bit audio; 0 is always silence).
`'MAC3'`	Samples have been compressed using MACE 3:1.
`'MAC6'`	Samples have been compressed using MACE 6:1.
`'ima4'`	Samples have been compressed using IMA 4:1.
`'µlaw'`	Samples have been compressed using µLaw 2:1.

The sound media handler also adds some of its own fields to the sample description. For more information about each of these fields, see *Inside Macintosh: QuickTime*.

Field descriptions

Version — A 16-bit integer that must be set to 0.

Revision level — A 16-bit integer that must be set to 0.

Vendor — A 32-bit integer that must be set to 0.

Number of channels — A 16-bit integer that indicates the number of sound channels used by the sound sample. Set this field to 1 for monaural sounds; set it to 2 for stereo sounds.

Sample size — A 16-bit integer that specifies the number of bits in each uncompressed sound sample. Set this field to 8 for 8-bit sound and

	to 16 for 16-bit sound. Sound stored in `'twos'` format may contain 32-bit samples.
Compression ID	A 16-bit integer that must be set to 0.
Packet size	A 16-bit integer that must be set to 0.
Sample rate	A 32-bit unsigned fixed-point number that indicates the rate at which the sound samples were obtained. This number should match the media's time scale.

Sound Sample Data

The format of data stored in sound samples is completely dependent on the type of the compressed data stored in the sound sample description. The following sections discuss each of the formats supported by QuickTime.

Uncompressed 8-bit Sound

Eight-bit audio may be stored in either twos-complement or offset-binary encodings. In either case, if the data is in stereo, the left and right channels are interleaved.

Uncompressed 16-bit Sound

Sixteen-bit audio may be stored in either twos-complement or offset-binary encodings. However, twos-complement encoding is preferred. In either case, if the data is in stereo, the left and right channels are interleaved.

MACE 3:1 and 6:1

Not currently documented by Apple Computer. These are 8-bit formats.

IMA 4:1

The IMA encoding scheme is based on a standard developed by the International Multimedia Association for pulse code modulation (PCM) audio compression. QuickTime uses a slight variation of the format to allow for random access. IMA is a 16-bit audio format.

µLaw 2:1

The µLaw encoding scheme is used on North American and Japanese phone systems, and is coming into use for voice data interchange, and in PBXs, voice-mail systems, and Internet Talk Radio (via MIME). In µLaw encoding, 14 bits of linear sample data are reduced to 8 bits of logarithmic data.

Time Code Media

Time code media is used to store time code data in QuickTime movies. It has a media type of `'tmcd'`.

Time Code Sample Description

The time code sample description contains information that defines how to interpret time code media data. This sample description is based on the standard sample description header, as described in "Sample Table Atoms" beginning on page 45.

The Data format field in the sample description is always set to `'tmcd'`.

The time code media handler also adds some of its own fields to the sample description.

Field descriptions

Reserved — A 32-bit integer that is reserved for future use. Set this field to 0.

Flags — A 32-bit integer containing flags that identify some time code characteristics. The following flags are defined.

- Drop frame — Indicates whether the time code is drop frame. Set it to 1 if the time code is drop frame. This flag's value is 0x0001.
- 24 hour max — Indicates whether the time code wraps after 24 hours. Set it to 1 if the time code wraps. This flag's value is 0x0002.
- Negative times OK — Indicates whether negative time values are allowed. Set it to 1 if the time code supports negative values. This flag's value is 0x0004.
- Counter — Indicates whether the time value corresponds to a tape counter value. Set it to 1 if the time code values are tape counter values. This flag's value is 0x0008.

Time scale — A 32-bit integer that specifies the time scale for interpreting the Frame duration field.

Frame duration — A 32-bit integer that indicates how long each frame lasts in real time.

Number of frames — An 8-bit integer that contains the number of frames per second for the time code format. If the time is a counter, this is the number of frames for each counter tick.

Reserved — A 24-bit quantity that must be set to 0.

Source reference — A user data atom containing information about the source video tape. The only currently used user data list entry is the `'name'` field. This entry contains an international text item specifying the name of the source tape.

Time Code Media Information Atom

The time code media also requires a media information atom. This atom contains information governing how the time code text is displayed. This media information atom is stored in a base media information atom (see "Base Media Information Atoms" beginning on page 38 for more information). The type of the time code media information atom is `'tcmi'`.

The time code media information atom contains the following data elements.

Field descriptions

Size	A 32-bit integer that specifies the number of bytes in this time code media information atom.
Type	A 32-bit integer that identifies the atom type; this field must be set to `'tcmi'`.
Version	A 1-byte specification of the version of this time code media information atom.
Flags	A 3-byte space for time code media information flags. Set this field to 0.
Text font	A 16-bit integer that indicates the font to use. Set this field to 0 to use the system font. If the Font name field contains a valid name, ignore this field.
Text face	A 16-bit integer that indicates the font's style. Set this field to 0 for normal text. You can enable other style options by using one or more of the following bit masks:

0x0001	Bold
0x0002	Italic
0x0004	Underline
0x0008	Outline
0x0010	Shadow
0x0020	Condense
0x0040	Extend

Text size	A 16-bit integer that specifies the point size of the time code text.
Text color	A 48-bit RGB color value for the time code text.
Background color	A 48-bit RGB background color for the time code text.
Font name	A Pascal string specifying the name of the time code text's font.

Time Code Sample Data

There are two different sample data formats used by time code media.

If the Counter flag is set to 1 in the time code sample description, the sample data is a counter value. Each sample contains a 32-bit integer counter value.

If the Counter flag is set to 0 in the time code sample description, the sample data format is a time code record, as follows.

Field descriptions

Hours	An 8-bit unsigned integer that indicates the starting number of hours.
Negative	A 1-bit value indicating the time's sign. If the bit is set to 1, the time code record value is negative.
Minutes	A 7-bit integer that contains the starting number of minutes.
Seconds	An 8-bit unsigned integer indicating the starting number of seconds.

Frames An 8-bit unsigned integer that specifies the starting number of frames. This field's value cannot exceed the value of the Number of frames field in the time code sample description.

Text Media

Text media is used to store text data in QuickTime movies. It has a media type of `'text'`.

Text Sample Description

The time code sample description contains information that defines how to interpret time code media data. This sample description is based on the standard sample description header, as described in "Sample Table Atoms" beginning on page 45.

The Data format field in the sample description is always set to `'text'`.

The text media handler also adds some of its own fields to the sample description.

Field descriptions

Display flags A 32-bit integer containing flags that describe how the text should be drawn. The following flags are defined.

Don't auto scale
: Controls text scaling. If this flag is set to 1, the text media handler reflows the text instead of scaling when the track is scaled. This flag's value is 0x0002.

Use movie background color
: Controls background color. If this flag is set to 1, the text media handler ignores the Background color field in the text sample description and uses the movie's background color instead. This flag's value is 0x0008.

Scroll in
: Controls text scrolling. If this flag is set to 1, the text media handler scrolls the text in until the last of the text is in view. This flag's value is 0x0020.

Scroll out
: Controls text scrolling. If this flag is set to 1, the text media handler scrolls the text out until the last of the text is gone. This flag's value is 0x0040.

Horizontal scroll
: Controls text scrolling. If this flag is set to 1, the text media handler scrolls the text horizontally; otherwise, it scrolls the text vertically. This flag's value is 0x0080.

Reverse scroll
: Controls text scrolling. If this flag is set to 1, the text media handler scrolls down (if scrolling vertically) or backward (if scrolling horizontally; horizontal scrolling also depends upon text justification). This flag's value is 0x0100.

	Continuous scroll	Controls text scrolling. If this flag is set to 1, the text media handler displays new samples by scrolling out the old ones. This flag's value is 0x0200.
	Drop shadow	Controls drop shadow. If this flag is set to1, the text media handler displays the text with a drop shadow. This flag's value is 0x1000.
	Anti-alias	Controls anti-aliasing. If this flag is set to 1, the text media handler uses anti-aliasing when drawing text. This flag's value is 0x2000.
	Key text	Controls background color. If this flag is set to 1, the text media handler does not display the background color, so that the text overlay background tracks. This flag's value is 0x4000.
Text justification	A 32-bit integer that indicates how the text should be aligned. Set this field to 0 for left-justified text, to 1 for centered text, and to –1 for right-justified text.	
Background color	A 48-bit RGB color that specifies the text's background color.	
Default text box	A 64-bit rectangle that specifies an area to receive text (top, left, bottom, right). Typically this field is set to all zeros.	
Reserved	A 64-bit value that must be set to 0.	
Font number	A 16-bit value that must be set to 0.	
Font face	A 16-bit integer that indicates the font's style. Set this field to 0 for normal text. You can enable other style options by using one or more of the following bit masks:	
	0x0001	Bold
	0x0002	Italic
	0x0004	Underline
	0x0008	Outline
	0x0010	Shadow
	0x0020	Condense
	0x0040	Extend
Reserved	An 8-bit value that must be set to 0.	
Reserved	A 16-bit value that must be set to 0.	
Foreground color	A 48-bit RGB color that specifies the text's foreground color.	
Text name	A Pascal string specifying the name of the font to use to display the text.	

Text Sample Data

The format of the text data is a 16-bit length followed by the actual text. The length word specifies the number of bytes of text, not including the length word itself. Following the text, there may be one or more atoms containing additional information for drawing and searching the text.

Table 1-9 lists the currently defined text sample extensions.

Table 1-9 Text sample extensions

Text sample extension	Description
`'styl'`	Style information for the text. Allows you to override the default style in the sample description or to define more than one style for a sample. The data is a TextEdit style scrap.
`'ftab'`	Table of font names. Each table entry contains a font number (stored in a 16-bit integer) and a font name (stored in a Pascal string). This atom is required if the `'styl'` atom is present.
`'hlit'`	Highlight information. The atom data consists of two 32-bit integers. The first contains the starting offset for the highlighted text, and the second has the ending offset. Highlight samples can be in key frames or in differenced frames. When it's used in a differenced frame, the sample should contain a zero-length piece of text.
`'hclr'`	Highlight color. This atom specifies the 48-bit RGB color to use for highlighting.
`'drpo'`	Drop shadow offset. When the Display flags indicate drop shadow style, this atom can be used to override the default drop shadow placement. The data consists of two 16-bit integers. The first indicates the horizontal displacement of the drop shadow, in pixels; the second, the vertical displacement.
`'drpt'`	Drop shadow transparency. The data is a 16-bit integer between 0 and 256 indicating the degree of transparency of the drop shadow. A value of 256 makes the drop shadow completely opaque.
`'imag'`	Image font data. This atom contains two more atoms. An `'idat'` atom contains compressed image data used to draw the text when the required fonts are not available. An `'idsc'` atom contains a video sample description describing the format of the compressed image data.
`'metr'`	Image font highlighting. This atom contains metric information that governs highlighting when an `'imag'` atom is used for drawing.

Music Media

Music media is used to store note-based audio data, such as MIDI data, in QuickTime movies. It has a media type of `'musi'`.

Music Sample Description

The music sample description uses the standard sample description header, as described in "Sample Table Atoms" beginning on page 45.

The data format field in the sample description is always set to `'musi'`. The music media handler adds an additional 32-bit integer field to the sample description containing flags. Currently no flags are defined, and this field should be set to 0.

Following the Flags field, there may be appended data in the QuickTime music format. This data consists of part-to-instrument mappings in the form of general events containing note requests. One note request event should be present for each part that will be used in the sample data.

Music Sample Data

The sample data for music samples consists entirely of data in the QuickTime music format. Typically, up to 30 seconds of notes will be grouped into a single sample. For a complete description of the QuickTime music format, see your most recent QuickTime developer update documentation.

MPEG Media

MPEG media is used to store MPEG streams in QuickTime movies. It has a media type of `'MPEG'`.

MPEG Sample Description

The MPEG sample description uses the standard sample description header, as described in “Sample Table Atoms” beginning on page 45.

The Data format field in the sample description is always set to `'MPEG'`. The MPEG media handler adds no additional fields to the sample description.

MPEG Sample Data

Each sample in an MPEG media is an entire MPEG stream. This can mean that a single MPEG sample may be several hundred megabytes in size. The MPEG encoding used by QuickTime corresponds to the ISO standard, as described in ISO document CD 11172.

Sprite Media

Sprite media is used to store character-based animation data in QuickTime movies. It has a media type of `'sprt'`.

Sprite Sample Description

The base sample description uses the standard sample description header, as described in “Sample Table Atoms” beginning on page 45.

The Data format field in the sample description is always set to `'sprt'`. The base media handler adds no additional fields to the sample description.

Sprite Sample Data

All sprite samples are stored in QT atom structures. The sprite media uses both key frames and differenced frames. The key frames contain all of the sprite's image data and the initial settings for each of the sprite's properties.

A key frame always contains a shared data atom of type `'dflt'`. This atom contains data to be shared between the sprites, consisting mainly of image data and sample descriptions. The shared data atom contains a single sprite image container atom, with an atom type value of `'imct'` and an ID value of 1.

The sprite image container atom stores one or more sprite image atoms of type `'imag'`. Each sprite image atom contains a video sample description immediately followed by the sprite's compressed image data. The sprite image atoms should have ID numbers starting at 1 and counting consecutively upward.

The key frame must also contain definitions for each sprite in atoms of type `'sprt'`. Sprite atoms should have ID numbers that start at 1 and count consecutively upward. Each sprite atom contains a list of properties. Four of the properties are required: Image index, Matrix, Layer, and Visibility. Table 1-10 shows all currently defined sprite properties.

Table 1-10 Sprite properties

Property Name	Property Value	Description
Matrix	1	Specifies a transformation matrix that describes the sprite's location, scaling, and so on.
Layer	5	Contains a 16-bit integer value specifying the layer into which the sprite is to be drawn (lower layers are rendered on top of higher layers)
Visibility	4	Contains a 16-bit integer that controls the sprite's visibility: set to 1 if the sprite is visible; 0 if not
Graphics mode	6	Contains a 32-bit integer specifying the sprite's graphics mode; this value is always followed by a 48-bit RGB value
Image index	100	Contains the atom ID of the sprite's image atom

The frame difference sample differs from the key frame sample in two ways. First, the frame difference sample does not contain a shared data atom. All shared data must appear in the key frame. Second, only those sprite properties that change need to be specified. If none of a sprite's properties change in a given frame, then the sprite does not need an atom in the differenced frame.

The frame difference sample can be used in one of two ways: the frame differences can be combined, as with video key frames, to construct the current frame; or the current frame can be derived by combining only the key frame and the current frame difference.

Base Media

Base media is used to store data that is not intended to be displayed in QuickTime movies. The data may be stored for an application to retrieve, or it may be used to modify another track. It has a media type of `'gnrc'`. Other media types are often built on top of the base media. Examples of media types derived from the base media include MPEG, sprite, text, music, and time code.

Base Sample Description

The base sample description uses the standard sample description header, as described in "Sample Table Atoms" beginning on page 45.

The Data format field in the sample description is always set to `'gnrc'`. The base media handler adds no additional fields to the sample description.

Base Sample Data

The base media handler defines no data formats. Applications may put whatever data they require into the samples.

Tween Media

Tween media is used to store pairs of values to be interpolated between in QuickTime movies. These interpolated values modify the playback of other media types by using track references and track input maps. For example, a tween media can generate gradually-changing relative volume levels to cause an audio track to fade out. It has a media type of `'twen'`.

Tween Sample Description

The tween sample description uses the standard sample description header, as described in "Sample Table Atoms" beginning on page 45.

The Data format field in the sample description is always set to `'twen'`. The tween media handler adds no additional fields to the sample description.

Tween Sample Data

Tween sample data is stored in QT atom structures.

At the root level, there are one or more tween entry atoms; these atoms have an atom type value of `'twen'`. Each tween entry atom completely describes one interpolation operation. These atoms should be consecutively numbered starting at 1, using the Atom ID field.

Each tween entry atom contains several more atoms that describe how to perform the interpolation. The Atom ID field in each of these atoms must be set to 1.

- Tween start atom (atom type is `'twst'`). This atom specifies the time at which the interpolation is to start. The time is expressed in the media's time coordinate system,

relative to the start of the media sample. If this atom is not present, the starting offset is assumed to be 0.

- Tween duration atom (atom type is `'twdu'`). This atom specifies how long the interpolation is to last. The time is expressed in the media's time coordinate system. If this atom is not present, the duration is assumed to be the length of the sample.
- Tween data atom (atom type is `'twdt'`). This atom contains the actual values for the interpolation. The contents depend on the value of the tween type atom.
- Tween type atom (atom type is `'twnt'`). Describes the type of interpolation to perform. Table 1-11 shows all currently defined tween types. All tween types are currently supported using linear interpolation.

Table 1-11 Tween type values

Tween type	Value	Tween data
16-bit integer	1	Two 16-bit integers.
32-bit integer	2	Two 32-bit integers.
32-bit fixed-point	3	Two 32-bit fixed-point numbers.
Point: two 16-bit integers	4	Two points.
Rectangle: four 16-bit integers	5	Two rectangles.
QuickDraw region	6	Two matrices. The tween entry atom must contain a `'qdrg'` atom with an Atom ID value of 1. The region is transformed through the interpolated matrices.
Matrix	7	Two matrices.
RGB color: three 16-bit integers	8	Two RGB colors.
Graphics mode with RGB color	9	Two graphics modes with RGB color. Only the RGB color is interpolated. The graphics modes must be the same.

3D Media

QuickTime movies store 3D image data in a base media. This media has a media type of `'qd3d'`.

3D Sample Description

The 3D sample description uses the standard sample description header, as described in "Sample Table Atoms" beginning on page 45.

The Data format field in the sample description is always set to `'qd3d'`. The 3D media handler adds no additional fields to the sample description.

3D Sample Data

The 3D samples are stored in the 3D metafile format developed for QuickDraw 3D. Please refer to *Inside Macintosh: 3D Metafile Reference* for details.

Basic Data Types

This section describes a number of common data types that are used in QuickTime files. For information about other QuickTime data types, refer to *Inside Macintosh: QuickTime*.

Language Code Values

Some elements of a QuickTime file may be associated with a particular spoken language. To indicate the language associated with a particular object, the QuickTime file format uses language codes from the Macintosh Script Manager. Table 1-12 lists the language codes supported by QuickTime.

Table 1-12 QuickTime language code values

Language	Value	Language	Value
English	0	Georgian	52
French	1	Moldavian	53
German	2	Kirghiz	54
Italian	3	Tajiki	55
Dutch	4	Turkmen	56
Swedish	5	Mongolian	57
Spanish	6	MongolianCyr	58
Danish	7	Pashto	59
Portuguese	8	Kurdish	60
Norwegian	9	Kashmiri	61
Hebrew	10	Sindhi	62
Japanese	11	Tibetan	63
Arabic	12	Nepali	64
Finnish	13	Sanskrit	65
Greek	14	Marathi	66
Icelandic	15	Bengali	67
Maltese	16	Assamese	68

Table 1-12 QuickTime language code values (continued)

Language	Value	Language	Value
Turkish	17	Gujarati	69
Croatian	18	Punjabi	70
Traditional Chinese	19	Oriya	71
Urdu	20	Malayalam	72
Hindi	21	Kannada	73
Thai	22	Tamil	74
Korean	23	Telugu	75
Lithuanian	24	Sinhalese	76
Polish	25	Burmese	77
Hungarian	26	Khmer	78
Estonian	27	Lao	79
Lettish	28	Vietnamese	80
Latvian	28	Indonesian	81
Saamisk	29	Tagalog	82
Lappish	29	MalayRoman	83
Faeroese	30	MalayArabic	84
Farsi	31	Amharic	85
Persian	31	Tigrinya	86
Russian	32	Galla	87
Simplified Chinese	33	Oromo	87
Flemish	34	Somali	88
Irish	35	Swahili	89
Albanian	36	Ruanda	90
Romanian	37	Rundi	91
Czech	38	Chewa	92
Slovak	39	Malagasy	93
Slovenian	40	Esperanto	94
Yiddish	41	Welsh	128
Serbian	42	Basque	129
Macedonian	43	Catalan	130
Bulgarian	44	Latin	131
Ukrainian	45	Quechua	132

Table 1-12 QuickTime language code values (continued)

Language	Value	Language	Value
Byelorussian	46	Guarani	133
Uzbek	47	Aymara	134
Kazakh	48	Tatar	135
Azerbaijani	49	Uighur	136
AzerbaijanAr	50	Dzongkha	137
Armenian	51	JavaneseRom	138
Georgian	52		

Calendar Date and Time Values

QuickTime movies store date and time information in Macintosh date format: a 32-bit value indicating the number of seconds that have passed since midnight, January 1, 1904.

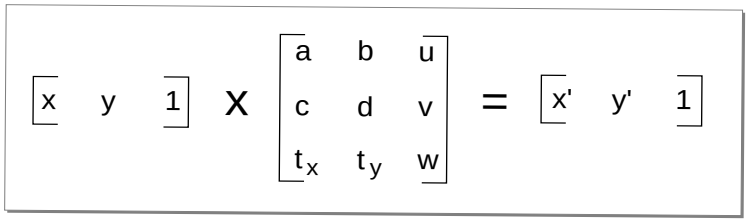
Matrices

QuickTime files use matrices to describe spatial information about many objects, such as tracks within a movie.

A transformation matrix defines how to map points from one coordinate space into another coordinate space. By modifying the contents of a transformation matrix, you can perform several standard graphics display operations, including translation, rotation, and scaling. The matrix used to accomplish two-dimensional transformations is described mathematically by a 3-by-3 matrix.

All values in the matrix are 32-bit fixed-point numbers divided as 16.16, except for the {u, v, w} column that contains 32-bit fixed-point numbers divided as 2.30. Figure 1-44 depicts how QuickTime uses matrices to transform displayed objects.

Figure 1-44 How display matrices are used in QuickTime



Graphics Modes

QuickTime files use graphics modes to describe how one video or graphics layer should be combined with the layers beneath it. Graphics modes are also known as transfer modes. Some graphics modes require a color to be specified for certain operations, such as blending to determine the blend level. QuickTime uses the graphics modes defined by QuickDraw. For information on QuickDraw graphics modes, see *Inside Macintosh: Imaging*.

The most common graphics modes are `srcCopy` (0x00) and `ditherCopy` (0x040), which simply indicate that the image should not blend with the image behind it, but overwrite it. QuickTime also defines several new graphics modes.

Table 1-13 lists the new graphics modes supported by QuickTime.

Table 1-13 QuickTime graphics modes

Graphics mode value	Description
0x0100	Straight alpha
0x0101	Alpha pre-multiplied with white
0x0102	Alpha pre-multiplied with black

RGB Colors

Many atoms in the QuickTime file format contain RGB color values. These are usually stored as three consecutive unsigned 16-bit integers in the following order: red, green, blue.

Examples

This section contains a number of examples that help you pull together all of the material in this book by examining the atom structure that results from a number of different scenarios. This section is divided into the following topics:

- "Creating Video Tracks at 30 Frames-per-second" discusses creating 30 fps video
- "Creating Video Tracks at 29.97 Frames-per-second" describes creating 29.97 fps video
- "Creating Audio Tracks at 44.1Khz" provides an example of creating an audio track
- "Creating a Time Code Track for 29.97 FPS Video" presents a time code track example
- "Playing With Edit Lists" discusses how to interpret edit list data
- "Interleaving Movie Data" shows how a movie's tracks are interleaved in the movie data file

- "Referencing Two Data Files With a Single Track" shows how track data may reside in more than one file

Creating Video Tracks at 30 Frames-per-second

The duration of a video frame is stored in the time-to-sample atom contained within a sample table atom. This duration cannot be interpreted without the media's time scale, which defines the units-per-second for the duration. In this example, each frame has the same duration, so the time-to-sample atom has one entry that applies to all video frames in the media.

As long as the ratio between frame duration and media time scale remains 1:30, any combination of values can be used for the duration and time scale. The larger the time scale the shorter the maximum duration. Since a movie defaults to a time scale of 600, this is a good number to use. It is also the least common multiple for 24, 25, and 30, making it handy for much of the math you are likely to encounter when making a movie.

The movie time scale is independent of the media time scale. Since you want to avoid movie edits that don't land on frame boundaries, it is a good idea to keep the movie time scale and the media time scale the same, or the movie time scale should be an even multiple of the media time scale. The movie time scale is stored in the movie header atom.

With a time scale of 600 in the media header atom, the time-to-sample atom would contain the following data values:

Atom size	24
Atom type	`'stts'`
Version/Flags	0
Number of entries	1
Sample count	n
Sample duration	20

Creating Video Tracks at 29.97 Frames-per-second

NTSC color video is not 30 frames-per-second (fps), but actually 29.97 fps. The previous example showed how the media time scale and the duration of the frames specify the video's frame rate. By setting the media's time scale to 2997 units per second and setting the frame durations to 100 units each, the effective rate is 29.97 fps exactly.

In this situation, it is also a good idea to set the movie time scale to 2997 in order to avoid movie edits that don't land on frame boundaries. The movie's time scale is stored in the movie header atom.

With a time scale of 2997 in the media header atom, the time-to-sample atom would contain the following data values:

Atom size	24
Atom type	`'stts'`
Version/Flags	0
Number of entries	1
Sample count	n
Sample duration	100

Creating Audio Tracks at 44.1Khz

The duration of an audio sample is stored in the time-to-sample atom contained in a sample table atom. This duration cannot be interpreted without the media's time scale, which defines the units-per-second for the duration. With audio, the duration of each audio sample is typically 1, so the time-to-sample atom has one entry that applies to all audio samples.

With a time scale of 44100 in the media header atom, the time-to-sample atom would contain the following data values:

Atom size	24
Atom type	`'stts'`
Version/Flags	0
Number of entries	1
Sample count	n
Sample duration	1

This atom does not indicate whether the audio is stereo or mono or whether it contains 8-bit or 16-bit samples. That information is stored in the sound sample description atom, which is contained in the sample table atom.

Creating a Time Code Track for 29.97 FPS Video

A time code track specifies time code information for other tracks. The time code keeps track of the time codes of the original source of the video and audio. After a movie has been edited, the time code can be extracted to determine the source tape and the time codes of the frames.

It is important that the time code track has the same time scale as the video track. Otherwise, the time code will not tick at the exact same time as the video track.

For each contiguous source tape segment, there is a single time code sample that specifies the time code value corresponding to the start of the segment. From this sample, the time code value can be determined for any point in the segment.

The sample description for a time code track specifies the time code system being used (for example, 30 fps drop-frame) and the source information. Each sample is a time code value.

Since the time code media handler is a derived from the base media handler, the media information atom starts with a generic media header atom. The time code atoms would contain the following data values:

Atom size	77		
Atom type	`'gmhd'`		
	Atom size	69	
	Atom type	`'gmin'`	
	Version/Flags	0	
	Graphics mode	0x0040	
	Opcolor (red)	0x8000	
	Opcolor (green)	0x8000	
	Opcolor (blue)	0x8000	
	Balance	0	
	Reserved	0	
	Atom size	45	
	Atom type	`'tmcd'`	
		Atom size	37
		Atom type	`'tcmi'`
		Version/Flags	0
		Text font	0 (system font)
		Text face	0 (plain)
		Text size	12
		Text color (red)	0
		Text color (green)	0
		Text color (blue)	0
		Background color (red)	0
		Background color (green)	0
		Background color (blue)	0
		Font name	`"\pChicago"` (Pascal string)

The sample table atom contains all the standard sample atoms and has the following data values:

Atom size	174		
Atom type	`'stbl'` (sample table)		
	Atom size	74	
	Atom type	`'stsd'` (sample description)	
	Version/Flags	0	
	Number of entries	1	
	Sample description size [1]	58	
	Data format [1]	`'tmcd'`	
	Reserved [1]	0	
	Data reference index [1]	1	
	Flags [1]	0	
	Flags (time code) [1]	7 (drop frame + 24 hour + negative times ok)	
	Time scale[1]	2997	
	Frame duration[1]	100	
	Number of frames[1]	20	
		Atom size	24
		Atom type	`'name'`
		String length	12
		Language code	0 (English)
		Name	“my tape name”
	Atom size	24	
	Atom type	`'stts'` (time-to-sample)	
	Version/Flags	0	
	Number of entries	1	
	Sample count[1]	1	
	Sample duration[1]	1	
	Atom size	28	
	Atom type	`'stsc'` (sample-to-chunk)	
	Version/Flags	0	
	Number of entries	1	
	First chunk[1]	1	
	Samples per chunk[1]	1	

Sample description ID[1]	1
Atom size	20
Atom type	`'stsz'` (sample size)
Version/Flags	0
Sample size	4
Number of entries	1
Atom size	20
Atom type	`'stco'` (chunk offset)
Version/Flags	0
Number of entries	1
Offset[1]	(offset into file of chunk 1)

In the example, let's assume that the segment's beginning time code is 1:15:32.4 (1 hour, 15 minutes, 32 seconds, and 4 frames). This time would be expressed in the data file as 0x010F2004 (0x01 = 1 hour; 0x0F = 15 minutes; 0x20 = 32 seconds; 0x04 = 4 frames).

The video and audio tracks must contain a track reference atom to indicate that they reference this time code track. The track reference is the same for both and is contained in the track atom (at the same level as the track header and media atoms).

This track reference would contain the following data values:

Atom size	12
Atom type	`'tref'`
Reference type	`'tmcd'`
Track ID of referenced track (time code track)	3

In this example, the video and sound tracks are tracks 1 and 2. The time code track is track 3.

Playing With Edit Lists

A segment of a movie can be repeated without duplicating media data by using edit lists. Suppose you have a single-track movie whose media time scale is 100 and track duration is 1000 (10 seconds). For this example the movie's time scale is 600. If there are no edits in the movie, the edit atom would contain the following data values:

Atom size	36	
Atom type	`'edts'`	
	Atom size	28
	Atom type	`'elst'`
	Version/Flags	0
	Number of entries	1
	Track duration	6000 (10 seconds)
	Media time	0
	Media rate	1.0

Because this is a single-track move, the track's duration in the track header atom is 6000, and the movie's duration in the movie header atom is 6000.

If you change the track to play the media from time 0 to time 2 seconds, and then play the media from time 0 to time 10 seconds, the edit atom would now contain these data values:

Atom size	48	
Atom type	`'edts'`	
	Atom size	40
	Atom type	`'elst'`
	Version/Flags	0
	Number of entries	2
	Track duration[1]	1200 (2 seconds)
	Media time[1]	0
	Media rate[1]	1.0
	Track duration[2]	6000 (10 seconds)
	Media time[2]	0
	Media rate[2]	1.0

Because the track is now 2 seconds longer, the track's duration in the track header atom must now be 7200, and the movie's duration in the movie header atom must also be 7200.

Currently, the media plays from time 0 to time 2, then plays from time 0 to time 10. If you take that repeated segment at the beginning (time 0 to time 2) and play it at double speed to maintain the original duration, the edit atom would now contain the following values:

Atom size	60	
Atom type	`'edts'`	
	Atom size	52
	Atom type	`'elst'`
	Version/Flags	0
	Number of entries	3
	Track duration[1]	600 (1 seconds)
	Media time[1]	0
	Media rate[1]	2.0
	Track duration[2]	600 (1 second)
	Media time[2]	0
	Media rate[2]	2.0
	Track duration[3]	4800 (8 seconds)
	Media time[3]	200
	Media rate[3]	1.0

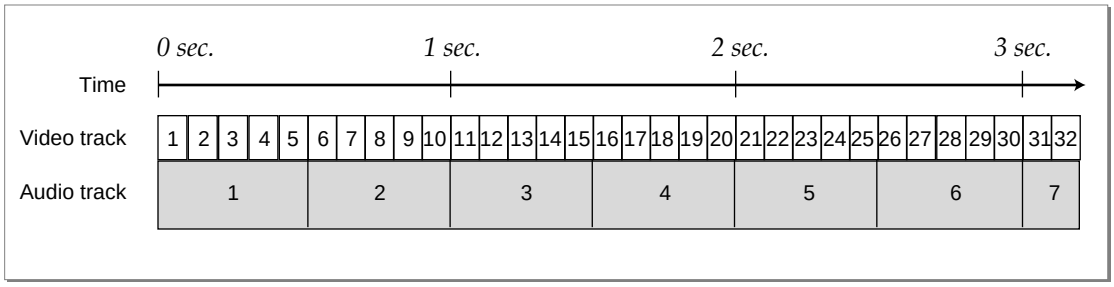
Because the track is now back to its original duration of 10 seconds, its duration in the track header atom is 6000 and the movie's duration in the movie header atom is 6000.

Interleaving Movie Data

In order to get optimal movie playback, you must create the movie with interleaved data. The data for the movie is placed on disk in time order so the video, sound, and other data for a particular time in the movie are close together in the file. This means that you will have to intersperse the data from different tracks. To illustrate this, consider a movie with a single video and a single audio track.

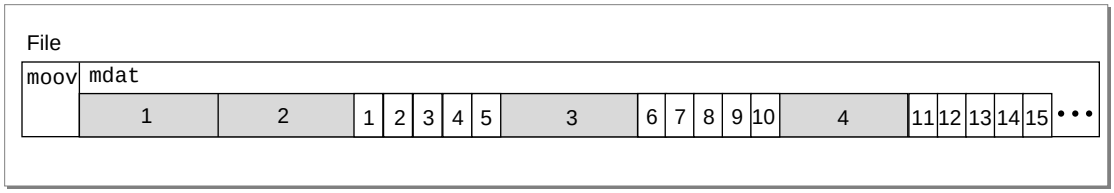
Figure 1-45 shows how the movie data was collected, and how the data would need to be played back for proper synchronization. In this example, the video data is recorded at 10 frames per second and the audio data is grouped into 1/2-second chunks.

Figure 1-45 Noninterleaved movie data



After the data has been interleaved on the disk, the movie data atom contains movie data in the order shown in Figure 1-46.

Figure 1-46 Interleaved movie data



In this example, the file begins with the movie atom (`'moov'`), followed by the movie data atom (`'mdat'`). In order to overcome any latencies in sound playback, at least one second of sound data is placed at the beginning of the interleaved data. This means that the sound and video data are offset from each other in the file by one second.

Referencing Two Data Files With a Single Track

The data reference index to be used for a given media sample is stored within that sample's sample description. Therefore, a track must contain multiple sample descriptions in order for that track to reference multiple data files. A different sample description must be used whenever the data file changes or whenever the format of the data changes. The sample-to-chunk atom determines which sample description to use for a sample.

The sample description atom would contain the following data values:

Atom size	…	
Atom type	`'stsd'`	
Version/Flags	0	
Number of entries	2	
	Sample description size[1]	…
	Data format	`'tmcd'`
	Reserved	0
	Data reference index	1
	(sample data)	…
	Sample description size[1]	…
	Data format	`'tmcd'`
	Reserved	0
	Data reference index	2
	(sample data)	…

If there were only 1 sample per chunk and the first 10 samples were extracted from sample description 2 and the next 30 samples were extracted from sample description 1, the sample-to-chunk atom would contain the following data values:

Atom size	40
Atom type	`'stsc'`
Version/Flags	0
Number of entries	2
First chunk[1]	1
Samples per chunk[1]	1
Sample description ID[1]	2
First chunk[2]	11
Samples per chunk[2]	1
Sample description ID[2]	1

The data reference atom would contain the following data values:

Atom size	…	
Atom type	`'dinf'`	
	Atom size	…
	Atom type	`'dref'`
	Version/Flags	0
	Number of entries	2
	Size[1]	…
	Type[1]	`'alis'`
	Version[1]	0
	Flags[1]	0 (not self referenced)
	Data reference[1]	[alias pointing to file #1]
	Size[2]	…
	Type[2]	`'alis'`
	Version[2]	0
	Flags[2]	0 (not self referenced)
	Data reference[2]	[alias pointing to file #2]

Index

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W, X

Y, Z

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Proof pages and final pages were created on an Apple LaserWriter Pro printer. Line art was created using Adobe™ Illustrator and Adobe Photoshop. PostScript™, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.

LEAD WRITER
Linda Kyrnitszke

WRITER
Doug Engfer

DEVELOPMENTAL EDITOR
Wendy Krafft

ILLUSTRATORS
Deb Dennis, Sandee Karr

PRODUCTION EDITOR
Gerri Gray

Special thanks to Peter Hoddie

RFC 1661 (RFC1661)

Internet RFC/STD/FYI/BCP Archives

RFC 1661 - The Point-to-Point Protocol (PPP)

Network Working Group W. Simpson, Editor
Request for Comments: 1661 Daydreamer
STD: 51 July 1994
Obsoletes: 1548
Category: Standards Track

The Point-to-Point Protocol (PPP)

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

The Point-to-Point Protocol (PPP) provides a standard method for transporting multi-protocol datagrams over point-to-point links. PPP is comprised of three main components:

1. A method for encapsulating multi-protocol datagrams.

2. A Link Control Protocol (LCP) for establishing, configuring, and testing the data-link connection.

3. A family of Network Control Protocols (NCPs) for establishing and configuring different network-layer protocols.

This document defines the PPP organization and methodology, and the PPP encapsulation, together with an extensible option negotiation mechanism which is able to negotiate a rich assortment of configuration parameters and provides additional management functions. The PPP Link Control Protocol (LCP) is described in terms of this mechanism.

Table of Contents

1. Introduction

The Point-to-Point Protocol is designed for simple links which transport packets between two peers. These links provide full-duplex simultaneous bi-directional operation, and are assumed to deliver packets in order. It is intended that PPP provide a common solution for easy connection of a wide variety of hosts, bridges and routers [1].

Encapsulation

The PPP encapsulation provides for multiplexing of different network-layer protocols simultaneously over the same link. The PPP encapsulation has been carefully designed to retain compatibility with most commonly used supporting hardware.

Only 8 additional octets are necessary to form the encapsulation when used within the default HDLC-like framing. In environments where bandwidth is at a premium, the encapsulation and framing may be shortened to 2 or 4 octets.

To support high speed implementations, the default encapsulation uses only simple fields, only one of which needs to be examined for demultiplexing. The default header and information fields fall on 32-bit boundaries, and the trailer may be padded to an arbitrary boundary.

Link Control Protocol

In order to be sufficiently versatile to be portable to a wide variety of environments, PPP provides a Link Control Protocol (LCP). The LCP is used to automatically agree upon the encapsulation format options, handle varying limits on sizes of packets, detect a looped-back link and other common misconfiguration errors, and terminate the link. Other optional facilities provided are authentication of the identity of its peer on the link, and determination when a link is functioning properly and when it is failing.

Network Control Protocols

Point-to-Point links tend to exacerbate many problems with the current family of network protocols. For instance, assignment and management of IP addresses, which is a problem even in LAN environments, is especially difficult over circuit-switched

point-to-point links (such as dial-up modem servers). These problems are handled by a family of Network Control Protocols (NCPs), which each manage the specific needs required by their

respective network-layer protocols. These NCPs are defined in companion documents.

Configuration

It is intended that PPP links be easy to configure. By design, the standard defaults handle all common configurations. The implementor can specify improvements to the default configuration, which are automatically communicated to the peer without operator intervention. Finally, the operator may explicitly configure options for the link which enable the link to operate in environments where it would otherwise be impossible.

This self-configuration is implemented through an extensible option negotiation mechanism, wherein each end of the link describes to the other its capabilities and requirements. Although the option negotiation mechanism described in this document is specified in terms of the Link Control Protocol (LCP), the same facilities are designed to be used by other control protocols, especially the family of NCPs.

1.1. Specification of Requirements

In this document, several words are used to signify the requirements of the specification. These words are often capitalized.

MUST — This word, or the adjective "required", means that the definition is an absolute requirement of the specification.

MUST NOT — This phrase means that the definition is an absolute prohibition 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 must be understood and carefully weighed before choosing a different course.

MAY — This word, or the adjective "optional", means that this item is one of an allowed set of alternatives. An implementation which does not include this option MUST be prepared to interoperate with another implementation which does include the option.

1.2. Terminology

This document frequently uses the following terms:

datagram — The unit of transmission in the network layer (such as IP). A datagram may be encapsulated in one or more packets passed to the data link layer.

frame — The unit of transmission at the data link layer. A frame may include a header and/or a trailer, along with some number of units of data.

packet — The basic unit of encapsulation, which is passed across the interface between the network layer and the data link layer. A packet is usually mapped to a frame; the exceptions are when data link layer fragmentation is being performed, or when multiple packets are incorporated into a single frame.

peer — The other end of the point-to-point link.

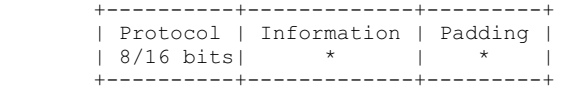
silently discard — The implementation discards the packet without further processing. The implementation SHOULD provide the capability of logging the error, including the contents of the silently discarded packet, and SHOULD record the event in a statistics counter.

2. PPP Encapsulation

The PPP encapsulation is used to disambiguate multiprotocol datagrams. This encapsulation requires framing to indicate the

beginning and end of the encapsulation. Methods of providing framing are specified in companion documents.

A summary of the PPP encapsulation is shown below. The fields are transmitted from left to right.



Protocol Field

The Protocol field is one or two octets, and its value identifies the datagram encapsulated in the Information field of the packet. The field is transmitted and received most significant octet first.

The structure of this field is consistent with the ISO 3309 extension mechanism for address fields. All Protocols MUST be odd; the least significant bit of the least significant octet MUST equal "1". Also, all Protocols MUST be assigned such that the least significant bit of the most significant octet equals "0". Frames received which don't comply with these rules MUST be treated as having an unrecognized Protocol.

Protocol field values in the "0***" to "3***" range identify the network-layer protocol of specific packets, and values in the "8***" to "b***" range identify packets belonging to the associated Network Control Protocols (NCPs), if any.

Protocol field values in the "4***" to "7***" range are used for protocols with low volume traffic which have no associated NCP. Protocol field values in the "c***" to "f***" range identify packets as link-layer Control Protocols (such as LCP).

Up-to-date values of the Protocol field are specified in the most recent "Assigned Numbers" RFC [2]. This specification reserves the following values:

Value (in hex)	Protocol Name
0001	Padding Protocol
0003 to 001f	reserved (transparency inefficient)
007d	reserved (Control Escape)
00cf	reserved (PPP NLPID)
00ff	reserved (compression inefficient)
8001 to 801f	unused
807d	unused
80cf	unused
80ff	unused
c021	Link Control Protocol
c023	Password Authentication Protocol
c025	Link Quality Report
c223	Challenge Handshake Authentication Protocol

Developers of new protocols MUST obtain a number from the Internet Assigned Numbers Authority (IANA), at IANA@isi.edu.

Information Field

The Information field is zero or more octets. The Information field contains the datagram for the protocol specified in the Protocol field.

The maximum length for the Information field, including Padding, but not including the Protocol field, is termed the Maximum Receive Unit (MRU), which defaults to 1500 octets. By negotiation, consenting PPP implementations may use other values for the MRU.

Padding

On transmission, the Information field MAY be padded with an arbitrary number of octets up to the MRU. It is the responsibility of each protocol to distinguish padding octets from real information.

3. PPP Link Operation

3.1. Overview

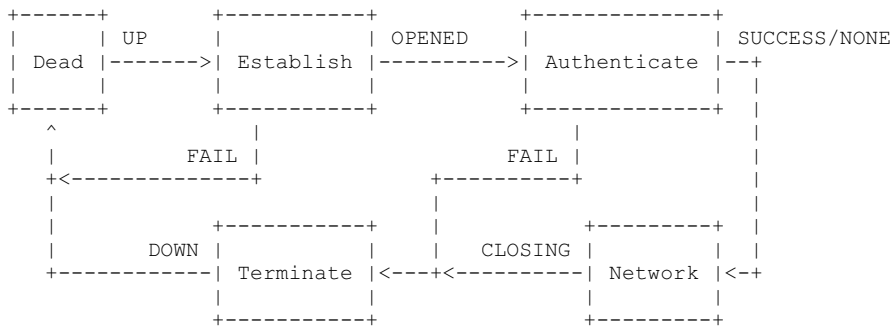
In order to establish communications over a point-to-point link, each end of the PPP link MUST first send LCP packets to configure and test the data link. After the link has been established, the peer MAY be authenticated.

Then, PPP MUST send NCP packets to choose and configure one or more network-layer protocols. Once each of the chosen network-layer protocols has been configured, datagrams from each network-layer protocol can be sent over the link.

The link will remain configured for communications until explicit LCP or NCP packets close the link down, or until some external event occurs (an inactivity timer expires or network administrator intervention).

3.2. Phase Diagram

In the process of configuring, maintaining and terminating the point-to-point link, the PPP link goes through several distinct phases which are specified in the following simplified state diagram:



Not all transitions are specified in this diagram. The following semantics MUST be followed.

3.3. Link Dead (physical-layer not ready)

The link necessarily begins and ends with this phase. When an external event (such as carrier detection or network administrator configuration) indicates that the physical-layer is ready to be used, PPP will proceed to the Link Establishment phase.

During this phase, the LCP automaton (described later) will be in the Initial or Starting states. The transition to the Link Establishment phase will signal an Up event to the LCP automaton.

Implementation Note:

> Typically, a link will return to this phase automatically after the disconnection of a modem. In the case of a hard-wired link, this phase may be extremely short -- merely long enough to detect the presence of the device.

3.4. Link Establishment Phase

The Link Control Protocol (LCP) is used to establish the connection through an exchange of Configure packets. This exchange is complete, and the LCP Opened state entered, once a Configure-Ack packet (described later) has been both sent and received.

All Configuration Options are assumed to be at default values unless altered by the configuration exchange. See the chapter on LCP Configuration Options for further discussion.

It is important to note that only Configuration Options which are independent of particular network-layer protocols are configured by LCP. Configuration of individual network-layer protocols is handled by separate Network Control Protocols (NCPs) during the Network-Layer Protocol phase.

Any non-LCP packets received during this phase MUST be silently discarded.

The receipt of the LCP Configure-Request causes a return to the Link Establishment phase from the Network-Layer Protocol phase or Authentication phase.

3.5. Authentication Phase

On some links it may be desirable to require a peer to authenticate itself before allowing network-layer protocol packets to be exchanged.

By default, authentication is not mandatory. If an implementation desires that the peer authenticate with some specific authentication protocol, then it MUST request the use of that authentication protocol during Link Establishment phase.

Authentication SHOULD take place as soon as possible after link establishment. However, link quality determination MAY occur concurrently. An implementation MUST NOT allow the exchange of link quality determination packets to delay authentication indefinitely.

Advancement from the Authentication phase to the Network-Layer Protocol phase MUST NOT occur until authentication has completed. If authentication fails, the authenticator SHOULD proceed instead to the Link Termination phase.

Only Link Control Protocol, authentication protocol, and link quality monitoring packets are allowed during this phase. All other packets received during this phase MUST be silently discarded.

Implementation Notes:

> An implementation SHOULD NOT fail authentication simply due to timeout or lack of response. The authentication SHOULD allow some method of retransmission, and proceed to the Link Termination phase only after a number of authentication attempts has been exceeded.
>
> The implementation responsible for commencing Link Termination phase is the implementation which has refused authentication to its peer.

3.6. Network-Layer Protocol Phase

Once PPP has finished the previous phases, each network-layer protocol (such as IP, IPX, or AppleTalk) MUST be separately configured by the appropriate Network Control Protocol (NCP).

Each NCP MAY be Opened and Closed at any time.

Implementation Note:

> Because an implementation may initially use a significant amount of time for link quality determination, implementations SHOULD avoid fixed timeouts when waiting for their peers to configure a NCP.

After a NCP has reached the Opened state, PPP will carry the corresponding network-layer protocol packets. Any supported network-layer protocol packets received when the corresponding NCP is not in the Opened state MUST be silently discarded.

Implementation Note:

> While LCP is in the Opened state, any protocol packet which is unsupported by the implementation MUST be returned in a Protocol-Reject (described later). Only protocols which are supported are silently discarded.

During this phase, link traffic consists of any possible combination of LCP, NCP, and network-layer protocol packets.

3.7. Link Termination Phase

PPP can terminate the link at any time. This might happen because of the loss of carrier, authentication failure, link quality failure, the expiration of an idle-period timer, or the administrative closing of the link.

LCP is used to close the link through an exchange of Terminate packets. When the link is closing, PPP informs the network-layer

protocols so that they may take appropriate action.

After the exchange of Terminate packets, the implementation SHOULD signal the physical-layer to disconnect in order to enforce the termination of the link, particularly in the case of an authentication failure. The sender of the Terminate-Request SHOULD disconnect after receiving a Terminate-Ack, or after the Restart counter expires. The receiver of a Terminate-Request SHOULD wait for the peer to disconnect, and MUST NOT disconnect until at least one Restart time has passed after sending a Terminate-Ack. PPP SHOULD proceed to the Link Dead phase.

Any non-LCP packets received during this phase MUST be silently discarded.

Implementation Note:

The closing of the link by LCP is sufficient. There is no need for each NCP to send a flurry of Terminate packets. Conversely, the fact that one NCP has Closed is not sufficient reason to cause the termination of the PPP link, even if that NCP was the only NCP currently in the Opened state.

4. The Option Negotiation Automaton

The finite-state automaton is defined by events, actions and state transitions. Events include reception of external commands such as Open and Close, expiration of the Restart timer, and reception of packets from a peer. Actions include the starting of the Restart timer and transmission of packets to the peer.

Some types of packets -- Configure-Naks and Configure-Rejects, or Code-Rejects and Protocol-Rejects, or Echo-Requests, Echo-Replies and Discard-Requests -- are not differentiated in the automaton descriptions. As will be described later, these packets do indeed serve different functions. However, they always cause the same transitions.

```
Events                                  Actions

Up   = lower layer is Up                tlu = This-Layer-Up
Down = lower layer is Down              tld = This-Layer-Down
Open = administrative Open              tls = This-Layer-Started
Close= administrative Close             tlf = This-Layer-Finished

TO+  = Timeout with counter > 0         irc = Initialize-Restart-Count
TO-  = Timeout with counter expired     zrc = Zero-Restart-Count

RCR+ = Receive-Configure-Request (Good) scr = Send-Configure-Request
RCR- = Receive-Configure-Request (Bad)
RCA  = Receive-Configure-Ack            sca = Send-Configure-Ack
RCN  = Receive-Configure-Nak/Rej        scn = Send-Configure-Nak/Rej

RTR  = Receive-Terminate-Request        str = Send-Terminate-Request
RTA  = Receive-Terminate-Ack            sta = Send-Terminate-Ack

RUC  = Receive-Unknown-Code             scj = Send-Code-Reject
RXJ+ = Receive-Code-Reject (permitted)
    or Receive-Protocol-Reject
RXJ- = Receive-Code-Reject (catastrophic)
    or Receive-Protocol-Reject
RXR  = Receive-Echo-Request             ser = Send-Echo-Reply
    or Receive-Echo-Reply
    or Receive-Discard-Request
```

4.1. State Transition Table

The complete state transition table follows. States are indicated horizontally, and events are read vertically. State transitions and actions are represented in the form action/new-state. Multiple actions are separated by commas, and may continue on succeeding lines as space requires; multiple actions may be implemented in any convenient order. The state may be followed by a letter, which indicates an explanatory footnote. The dash ('-') indicates an illegal transition.

```
      | State
      |   0        1         2         3         4         5
Events| Initial   Starting  Closed    Stopped   Closing   Stopping
------+-----------------------------------------------------------
```

Up	2	irc,scr/6	-	-	-	-
Down	-	-	0	tls/1	0	1
Open	tls/1	1	irc,scr/6	3r	5r	5r
Close	0	tlf/0	2	2	4	4
TO+	-	-	-	-	str/4	str/5
TO-	-	-	-	-	tlf/2	tlf/3
RCR+	-	-	sta/2	irc,scr,sca/8	4	5
RCR-	-	-	sta/2	irc,scr,scn/6	4	5
RCA	-	-	sta/2	sta/3	4	5
RCN	-	-	sta/2	sta/3	4	5
RTR	-	-	sta/2	sta/3	sta/4	sta/5
RTA	-	-	2	3	tlf/2	tlf/3
RUC	-	-	scj/2	scj/3	scj/4	scj/5
RXJ+	-	-	2	3	4	5
RXJ-	-	-	tlf/2	tlf/3	tlf/2	tlf/3
RXR	-	-	2	3	4	5

Events	State 6 Req-Sent	7 Ack-Rcvd	8 Ack-Sent	9 Opened
Up	-	-	-	-
Down	1	1	1	tld/1
Open	6	7	8	9r
Close	irc,str/4	irc,str/4	irc,str/4	tld,irc,str/4
TO+	scr/6	scr/6	scr/8	-
TO-	tlf/3p	tlf/3p	tlf/3p	-
RCR+	sca/8	sca,tlu/9	sca/8	tld,scr,sca/8
RCR-	scn/6	scn/7	scn/6	tld,scr,scn/6
RCA	irc/7	scr/6x	irc,tlu/9	tld,scr/6x
RCN	irc,scr/6	scr/6x	irc,scr/8	tld,scr/6x
RTR	sta/6	sta/6	sta/6	tld,zrc,sta/5
RTA	6	6	8	tld,scr/6
RUC	scj/6	scj/7	scj/8	scj/9
RXJ+	6	6	8	9
RXJ-	tlf/3	tlf/3	tlf/3	tld,irc,str/5
RXR	6	7	8	ser/9

The states in which the Restart timer is running are identifiable by the presence of TO events. Only the Send-Configure-Request, Send-Terminate-Request and Zero-Restart-Count actions start or re-start the Restart timer. The Restart timer is stopped when transitioning from any state where the timer is running to a state where the timer is not running.

The events and actions are defined according to a message passing architecture, rather than a signalling architecture. If an action is desired to control specific signals (such as DTR), additional actions are likely to be required.

[p] Passive option; see Stopped state discussion.

[r] Restart option; see Open event discussion.

[x] Crossed connection; see RCA event discussion.

4.2. States

Following is a more detailed description of each automaton state.

Initial

In the Initial state, the lower layer is unavailable (Down), and no Open has occurred. The Restart timer is not running in the Initial state.

Starting

The Starting state is the Open counterpart to the Initial state. An administrative Open has been initiated, but the lower layer is

still unavailable (Down). The Restart timer is not running in the Starting state.

When the lower layer becomes available (Up), a Configure-Request is sent.

Closed

In the Closed state, the link is available (Up), but no Open has occurred. The Restart timer is not running in the Closed state.

Upon reception of Configure-Request packets, a Terminate-Ack is sent. Terminate-Acks are silently discarded to avoid creating a loop.

Stopped

The Stopped state is the Open counterpart to the Closed state. It is entered when the automaton is waiting for a Down event after the This-Layer-Finished action, or after sending a Terminate-Ack. The Restart timer is not running in the Stopped state.

Upon reception of Configure-Request packets, an appropriate response is sent. Upon reception of other packets, a Terminate-Ack is sent. Terminate-Acks are silently discarded to avoid creating a loop.

Rationale:

The Stopped state is a junction state for link termination, link configuration failure, and other automaton failure modes. These potentially separate states have been combined.

There is a race condition between the Down event response (from

the This-Layer-Finished action) and the Receive-Configure-Request event. When a Configure-Request arrives before the Down event, the Down event will supercede by returning the automaton to the Starting state. This prevents attack by repetition.

Implementation Option:

After the peer fails to respond to Configure-Requests, an implementation MAY wait passively for the peer to send Configure-Requests. In this case, the This-Layer-Finished action is not used for the TO- event in states Req-Sent, Ack-Rcvd and Ack-Sent.

This option is useful for dedicated circuits, or circuits which have no status signals available, but SHOULD NOT be used for switched circuits.

Closing

In the Closing state, an attempt is made to terminate the connection. A Terminate-Request has been sent and the Restart timer is running, but a Terminate-Ack has not yet been received.

Upon reception of a Terminate-Ack, the Closed state is entered. Upon the expiration of the Restart timer, a new Terminate-Request is transmitted, and the Restart timer is restarted. After the Restart timer has expired Max-Terminate times, the Closed state is entered.

Stopping

The Stopping state is the Open counterpart to the Closing state. A Terminate-Request has been sent and the Restart timer is running, but a Terminate-Ack has not yet been received.

Rationale:

The Stopping state provides a well defined opportunity to terminate a link before allowing new traffic. After the link has terminated, a new configuration may occur via the Stopped or Starting states.

Request-Sent

In the Request-Sent state an attempt is made to configure the

connection. A Configure-Request has been sent and the Restart timer is running, but a Configure-Ack has not yet been received

nor has one been sent.

Ack-Received

In the Ack-Received state, a Configure-Request has been sent and a Configure-Ack has been received. The Restart timer is still running, since a Configure-Ack has not yet been sent.

Ack-Sent

In the Ack-Sent state, a Configure-Request and a Configure-Ack have both been sent, but a Configure-Ack has not yet been received. The Restart timer is running, since a Configure-Ack has not yet been received.

Opened

In the Opened state, a Configure-Ack has been both sent and received. The Restart timer is not running.

When entering the Opened state, the implementation SHOULD signal the upper layers that it is now Up. Conversely, when leaving the Opened state, the implementation SHOULD signal the upper layers that it is now Down.

4.3. Events

Transitions and actions in the automaton are caused by events.

Up

This event occurs when a lower layer indicates that it is ready to carry packets.

Typically, this event is used by a modem handling or calling process, or by some other coupling of the PPP link to the physical media, to signal LCP that the link is entering Link Establishment phase.

It also can be used by LCP to signal each NCP that the link is entering Network-Layer Protocol phase. That is, the This-Layer-Up action from LCP triggers the Up event in the NCP.

Down

This event occurs when a lower layer indicates that it is no

longer ready to carry packets.

Typically, this event is used by a modem handling or calling process, or by some other coupling of the PPP link to the physical media, to signal LCP that the link is entering Link Dead phase.

It also can be used by LCP to signal each NCP that the link is leaving Network-Layer Protocol phase. That is, the This-Layer-Down action from LCP triggers the Down event in the NCP.

Open

This event indicates that the link is administratively available for traffic; that is, the network administrator (human or program) has indicated that the link is allowed to be Opened. When this event occurs, and the link is not in the Opened state, the automaton attempts to send configuration packets to the peer.

If the automaton is not able to begin configuration (the lower layer is Down, or a previous Close event has not completed), the establishment of the link is automatically delayed.

When a Terminate-Request is received, or other events occur which cause the link to become unavailable, the automaton will progress to a state where the link is ready to re-open. No additional administrative intervention is necessary.

Implementation Option:

Experience has shown that users will execute an additional Open command when they want to renegotiate the link. This might

indicate that new values are to be negotiated.

Since this is not the meaning of the Open event, it is suggested that when an Open user command is executed in the Opened, Closing, Stopping, or Stopped states, the implementation issue a Down event, immediately followed by an Up event. Care must be taken that an intervening Down event cannot occur from another source.

The Down followed by an Up will cause an orderly renegotiation of the link, by progressing through the Starting to the Request-Sent state. This will cause the renegotiation of the link, without any harmful side effects.

Close

This event indicates that the link is not available for traffic;

that is, the network administrator (human or program) has indicated that the link is not allowed to be Opened. When this event occurs, and the link is not in the Closed state, the automaton attempts to terminate the connection. Futher attempts to re-configure the link are denied until a new Open event occurs.

Implementation Note:

When authentication fails, the link SHOULD be terminated, to prevent attack by repetition and denial of service to other users. Since the link is administratively available (by definition), this can be accomplished by simulating a Close event to the LCP, immediately followed by an Open event. Care must be taken that an intervening Close event cannot occur from another source.

The Close followed by an Open will cause an orderly termination of the link, by progressing through the Closing to the Stopping state, and the This-Layer-Finished action can disconnect the link. The automaton waits in the Stopped or Starting states for the next connection attempt.

Timeout (TO+,TO-)

This event indicates the expiration of the Restart timer. The Restart timer is used to time responses to Configure-Request and Terminate-Request packets.

The TO+ event indicates that the Restart counter continues to be greater than zero, which triggers the corresponding Configure-Request or Terminate-Request packet to be retransmitted.

The TO- event indicates that the Restart counter is not greater than zero, and no more packets need to be retransmitted.

Receive-Configure-Request (RCR+,RCR-)

This event occurs when a Configure-Request packet is received from the peer. The Configure-Request packet indicates the desire to open a connection and may specify Configuration Options. The Configure-Request packet is more fully described in a later section.

The RCR+ event indicates that the Configure-Request was acceptable, and triggers the transmission of a corresponding Configure-Ack.

The RCR- event indicates that the Configure-Request was

unacceptable, and triggers the transmission of a corresponding Configure-Nak or Configure-Reject.

Implementation Note:

These events may occur on a connection which is already in the Opened state. The implementation MUST be prepared to immediately renegotiate the Configuration Options.

Receive-Configure-Ack (RCA)

This event occurs when a valid Configure-Ack packet is received from the peer. The Configure-Ack packet is a positive response to

a Configure-Request packet. An out of sequence or otherwise invalid packet is silently discarded.

Implementation Note:

Since the correct packet has already been received before reaching the Ack-Rcvd or Opened states, it is extremely unlikely that another such packet will arrive. As specified, all invalid Ack/Nak/Rej packets are silently discarded, and do not affect the transitions of the automaton.

However, it is not impossible that a correctly formed packet will arrive through a coincidentally-timed cross-connection. It is more likely to be the result of an implementation error. At the very least, this occurance SHOULD be logged.

Receive-Configure-Nak/Rej (RCN)

This event occurs when a valid Configure-Nak or Configure-Reject packet is received from the peer. The Configure-Nak and Configure-Reject packets are negative responses to a Configure-Request packet. An out of sequence or otherwise invalid packet is silently discarded.

Implementation Note:

Although the Configure-Nak and Configure-Reject cause the same state transition in the automaton, these packets have significantly different effects on the Configuration Options sent in the resulting Configure-Request packet.

Receive-Terminate-Request (RTR)

This event occurs when a Terminate-Request packet is received. The Terminate-Request packet indicates the desire of the peer to

close the connection.

Implementation Note:

This event is not identical to the Close event (see above), and does not override the Open commands of the local network administrator. The implementation MUST be prepared to receive a new Configure-Request without network administrator intervention.

Receive-Terminate-Ack (RTA)

This event occurs when a Terminate-Ack packet is received from the peer. The Terminate-Ack packet is usually a response to a Terminate-Request packet. The Terminate-Ack packet may also indicate that the peer is in Closed or Stopped states, and serves to re-synchronize the link configuration.

Receive-Unknown-Code (RUC)

This event occurs when an un-interpretable packet is received from the peer. A Code-Reject packet is sent in response.

Receive-Code-Reject, Receive-Protocol-Reject (RXJ+,RXJ-)

This event occurs when a Code-Reject or a Protocol-Reject packet is received from the peer.

The RXJ+ event arises when the rejected value is acceptable, such as a Code-Reject of an extended code, or a Protocol-Reject of a NCP. These are within the scope of normal operation. The implementation MUST stop sending the offending packet type.

The RXJ- event arises when the rejected value is catastrophic, such as a Code-Reject of Configure-Request, or a Protocol-Reject of LCP! This event communicates an unrecoverable error that terminates the connection.

Receive-Echo-Request, Receive-Echo-Reply, Receive-Discard-Request (RXR)

This event occurs when an Echo-Request, Echo-Reply or Discard-Request packet is received from the peer. The Echo-Reply packet is a response to an Echo-Request packet. There is no reply to an Echo-Reply or Discard-Request packet.

4.4. Actions

Actions in the automaton are caused by events and typically indicate the transmission of packets and/or the starting or stopping of the Restart timer.

Illegal-Event (-)

This indicates an event that cannot occur in a properly implemented automaton. The implementation has an internal error, which should be reported and logged. No transition is taken, and the implementation SHOULD NOT reset or freeze.

This-Layer-Up (tlu)

This action indicates to the upper layers that the automaton is entering the Opened state.

Typically, this action is used by the LCP to signal the Up event to a NCP, Authentication Protocol, or Link Quality Protocol, or MAY be used by a NCP to indicate that the link is available for its network layer traffic.

This-Layer-Down (tld)

This action indicates to the upper layers that the automaton is leaving the Opened state.

Typically, this action is used by the LCP to signal the Down event to a NCP, Authentication Protocol, or Link Quality Protocol, or MAY be used by a NCP to indicate that the link is no longer available for its network layer traffic.

This-Layer-Started (tls)

This action indicates to the lower layers that the automaton is entering the Starting state, and the lower layer is needed for the link. The lower layer SHOULD respond with an Up event when the lower layer is available.

This results of this action are highly implementation dependent.

This-Layer-Finished (tlf)

This action indicates to the lower layers that the automaton is entering the Initial, Closed or Stopped states, and the lower layer is no longer needed for the link. The lower layer SHOULD respond with a Down event when the lower layer has terminated.

Typically, this action MAY be used by the LCP to advance to the Link Dead phase, or MAY be used by a NCP to indicate to the LCP that the link may terminate when there are no other NCPs open.

This results of this action are highly implementation dependent.

Initialize-Restart-Count (irc)

This action sets the Restart counter to the appropriate value (Max-Terminate or Max-Configure). The counter is decremented for each transmission, including the first.

Implementation Note:

In addition to setting the Restart counter, the implementation MUST set the timeout period to the initial value when Restart timer backoff is used.

Zero-Restart-Count (zrc)

This action sets the Restart counter to zero.

Implementation Note:

This action enables the FSA to pause before proceeding to the desired final state, allowing traffic to be processed by the peer. In addition to zeroing the Restart counter, the implementation MUST set the timeout period to an appropriate value.

Send-Configure-Request (scr)

A Configure-Request packet is transmitted. This indicates the desire to open a connection with a specified set of Configuration Options. The Restart timer is started when the Configure-Request packet is transmitted, to guard against packet loss. The Restart counter is decremented each time a Configure-Request is sent.

Send-Configure-Ack (sca)

A Configure-Ack packet is transmitted. This acknowledges the reception of a Configure-Request packet with an acceptable set of Configuration Options.

Send-Configure-Nak (scn)

A Configure-Nak or Configure-Reject packet is transmitted, as appropriate. This negative response reports the reception of a

Configure-Request packet with an unacceptable set of Configuration Options.

Configure-Nak packets are used to refuse a Configuration Option value, and to suggest a new, acceptable value. Configure-Reject packets are used to refuse all negotiation about a Configuration Option, typically because it is not recognized or implemented. The use of Configure-Nak versus Configure-Reject is more fully described in the chapter on LCP Packet Formats.

Send-Terminate-Request (str)

A Terminate-Request packet is transmitted. This indicates the desire to close a connection. The Restart timer is started when the Terminate-Request packet is transmitted, to guard against packet loss. The Restart counter is decremented each time a Terminate-Request is sent.

Send-Terminate-Ack (sta)

A Terminate-Ack packet is transmitted. This acknowledges the reception of a Terminate-Request packet or otherwise serves to synchronize the automatons.

Send-Code-Reject (scj)

A Code-Reject packet is transmitted. This indicates the reception of an unknown type of packet.

Send-Echo-Reply (ser)

An Echo-Reply packet is transmitted. This acknowledges the reception of an Echo-Request packet.

4.5. Loop Avoidance

The protocol makes a reasonable attempt at avoiding Configuration Option negotiation loops. However, the protocol does NOT guarantee that loops will not happen. As with any negotiation, it is possible to configure two PPP implementations with conflicting policies that will never converge. It is also possible to configure policies which do converge, but which take significant time to do so. Implementors should keep this in mind and SHOULD implement loop detection mechanisms or higher level timeouts.

4.6. Counters and Timers

Restart Timer

There is one special timer used by the automaton. The Restart timer is used to time transmissions of Configure-Request and Terminate-Request packets. Expiration of the Restart timer causes a Timeout event, and retransmission of the corresponding Configure-Request or Terminate-Request packet. The Restart timer MUST be configurable, but SHOULD default to three (3) seconds.

Implementation Note:

The Restart timer SHOULD be based on the speed of the link. The default value is designed for low speed (2,400 to 9,600 bps), high switching latency links (typical telephone lines). Higher speed links, or links with low switching latency, SHOULD have correspondingly faster retransmission times.

Instead of a constant value, the Restart timer MAY begin at an initial small value and increase to the configured final value. Each successive value less than the final value SHOULD be at least twice the previous value. The initial value SHOULD be large enough to account for the size of the packets, twice the round trip time for transmission at the link speed, and at least an additional 100 milliseconds to allow the peer to process the packets before responding. Some circuits add another 200 milliseconds of satellite delay. Round trip times for modems operating at 14,400 bps have been measured in the range of 160 to more than 600 milliseconds.

Max-Terminate

There is one required restart counter for Terminate-Requests. Max-Terminate indicates the number of Terminate-Request packets sent without receiving a Terminate-Ack before assuming that the peer is unable to respond. Max-Terminate MUST be configurable, but SHOULD default to two (2) transmissions.

Max-Configure

A similar counter is recommended for Configure-Requests. Max-Configure indicates the number of Configure-Request packets sent without receiving a valid Configure-Ack, Configure-Nak or Configure-Reject before assuming that the peer is unable to respond. Max-Configure MUST be configurable, but SHOULD default to ten (10) transmissions.

Max-Failure

A related counter is recommended for Configure-Nak. Max-Failure indicates the number of Configure-Nak packets sent without sending a Configure-Ack before assuming that configuration is not converging. Any further Configure-Nak packets for peer requested options are converted to Configure-Reject packets, and locally desired options are no longer appended. Max-Failure MUST be configurable, but SHOULD default to five (5) transmissions.

5. LCP Packet Formats

There are three classes of LCP packets:

1. Link Configuration packets used to establish and configure a link (Configure-Request, Configure-Ack, Configure-Nak and Configure-Reject).

2. Link Termination packets used to terminate a link (Terminate-Request and Terminate-Ack).

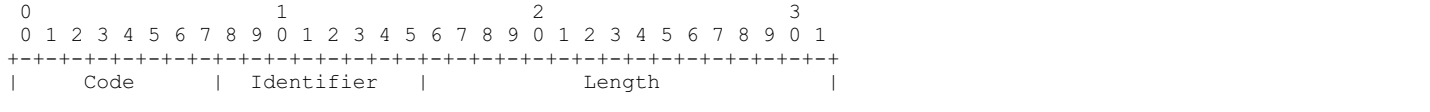
3. Link Maintenance packets used to manage and debug a link (Code-Reject, Protocol-Reject, Echo-Request, Echo-Reply, and Discard-Request).

In the interest of simplicity, there is no version field in the LCP packet. A correctly functioning LCP implementation will always respond to unknown Protocols and Codes with an easily recognizable LCP packet, thus providing a deterministic fallback mechanism for implementations of other versions.

Regardless of which Configuration Options are enabled, all LCP Link Configuration, Link Termination, and Code-Reject packets (codes 1 through 7) are always sent as if no Configuration Options were negotiated. In particular, each Configuration Option specifies a default value. This ensures that such LCP packets are always recognizable, even when one end of the link mistakenly believes the link to be open.

Exactly one LCP packet is encapsulated in the PPP Information field, where the PPP Protocol field indicates type hex c021 (Link Control Protocol).

A summary of the Link Control Protocol packet format is shown below. The fields are transmitted from left to right.



```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Data ...
+-+-+-+-+
```

Code

The Code field is one octet, and identifies the kind of LCP

packet. When a packet is received with an unknown Code field, a Code-Reject packet is transmitted.

Up-to-date values of the LCP Code field are specified in the most recent "Assigned Numbers" RFC [2]. This document concerns the following values:

1	Configure-Request
2	Configure-Ack
3	Configure-Nak
4	Configure-Reject
5	Terminate-Request
6	Terminate-Ack
7	Code-Reject
8	Protocol-Reject
9	Echo-Request
10	Echo-Reply
11	Discard-Request

Identifier

The Identifier field is one octet, and aids in matching requests and replies. When a packet is received with an invalid Identifier field, the packet is silently discarded without affecting the automaton.

Length

The Length field is two octets, and indicates the length of the LCP packet, including the Code, Identifier, Length and Data fields. The Length MUST NOT exceed the MRU of the link.

Octets outside the range of the Length field are treated as padding and are ignored on reception. When a packet is received with an invalid Length field, the packet is silently discarded without affecting the automaton.

Data

The Data field is zero or more octets, as indicated by the Length field. The format of the Data field is determined by the Code field.

5.1. Configure-Request

Description

An implementation wishing to open a connection MUST transmit a Configure-Request. The Options field is filled with any desired changes to the link defaults. Configuration Options SHOULD NOT be included with default values.

Upon reception of a Configure-Request, an appropriate reply MUST be transmitted.

A summary of the Configure-Request packet format is shown below. The fields are transmitted from left to right.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Options ...
+-+-+-+-+
```

Code

1 for Configure-Request.

Identifier

The Identifier field MUST be changed whenever the contents of the

Options field changes, and whenever a valid reply has been received for a previous request. For retransmissions, the Identifier MAY remain unchanged.

Options

The options field is variable in length, and contains the list of zero or more Configuration Options that the sender desires to negotiate. All Configuration Options are always negotiated simultaneously. The format of Configuration Options is further described in a later chapter.

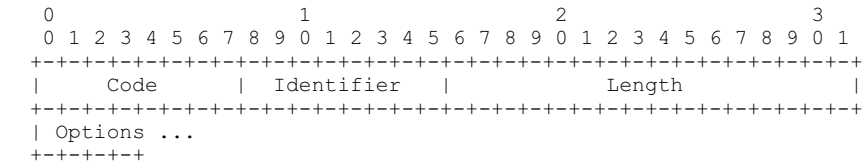
5.2. Configure-Ack

Description

If every Configuration Option received in a Configure-Request is recognizable and all values are acceptable, then the implementation MUST transmit a Configure-Ack. The acknowledged Configuration Options MUST NOT be reordered or modified in any way.

On reception of a Configure-Ack, the Identifier field MUST match that of the last transmitted Configure-Request. Additionally, the Configuration Options in a Configure-Ack MUST exactly match those of the last transmitted Configure-Request. Invalid packets are silently discarded.

A summary of the Configure-Ack packet format is shown below. The fields are transmitted from left to right.



Code

2 for Configure-Ack.

Identifier

The Identifier field is a copy of the Identifier field of the Configure-Request which caused this Configure-Ack.

Options

The Options field is variable in length, and contains the list of zero or more Configuration Options that the sender is acknowledging. All Configuration Options are always acknowledged simultaneously.

5.3. Configure-Nak

Description

If every instance of the received Configuration Options is recognizable, but some values are not acceptable, then the implementation MUST transmit a Configure-Nak. The Options field is filled with only the unacceptable Configuration Options from the Configure-Request. All acceptable Configuration Options are filtered out of the Configure-Nak, but otherwise the Configuration Options from the Configure-Request MUST NOT be reordered.

Options which have no value fields (boolean options) MUST use the Configure-Reject reply instead.

Each Configuration Option which is allowed only a single instance MUST be modified to a value acceptable to the Configure-Nak sender. The default value MAY be used, when this differs from the requested value.

When a particular type of Configuration Option can be listed more than once with different values, the Configure-Nak MUST include a list of all values for that option which are acceptable to the Configure-Nak sender. This includes acceptable values that were present in the Configure-Request.

Finally, an implementation may be configured to request the negotiation of a specific Configuration Option. If that option is not listed, then that option MAY be appended to the list of Nak'd Configuration Options, in order to prompt the peer to include that option in its next Configure-Request packet. Any value fields for the option MUST indicate values acceptable to the Configure-Nak sender.

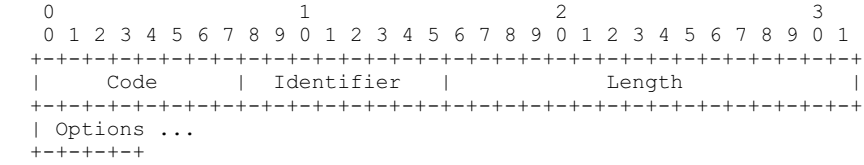
On reception of a Configure-Nak, the Identifier field MUST match that of the last transmitted Configure-Request. Invalid packets are silently discarded.

Reception of a valid Configure-Nak indicates that when a new Configure-Request is sent, the Configuration Options MAY be modified as specified in the Configure-Nak. When multiple instances of a Configuration Option are present, the peer SHOULD select a single value to include in its next Configure-Request packet.

Some Configuration Options have a variable length. Since the Nak'd Option has been modified by the peer, the implementation MUST be able to handle an Option length which is different from

the original Configure-Request.

A summary of the Configure-Nak packet format is shown below. The fields are transmitted from left to right.



Code

3 for Configure-Nak.

Identifier

The Identifier field is a copy of the Identifier field of the Configure-Request which caused this Configure-Nak.

Options

The Options field is variable in length, and contains the list of zero or more Configuration Options that the sender is Nak'ing. All Configuration Options are always Nak'd simultaneously.

5.4. Configure-Reject

Description

If some Configuration Options received in a Configure-Request are not recognizable or are not acceptable for negotiation (as configured by a network administrator), then the implementation MUST transmit a Configure-Reject. The Options field is filled with only the unacceptable Configuration Options from the Configure-Request. All recognizable and negotiable Configuration Options are filtered out of the Configure-Reject, but otherwise the Configuration Options MUST NOT be reordered or modified in any way.

On reception of a Configure-Reject, the Identifier field MUST match that of the last transmitted Configure-Request. Additionally, the Configuration Options in a Configure-Reject MUST

be a proper subset of those in the last transmitted Configure-Request. Invalid packets are silently discarded.

Reception of a valid Configure-Reject indicates that when a new Configure-Request is sent, it MUST NOT include any of the Configuration Options listed in the Configure-Reject.

A summary of the Configure-Reject packet format is shown below. The fields are transmitted from left to right.

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Options ...
+-+-+-+-+
```

Code

4 for Configure-Reject.

Identifier

The Identifier field is a copy of the Identifier field of the Configure-Request which caused this Configure-Reject.

Options

The Options field is variable in length, and contains the list of zero or more Configuration Options that the sender is rejecting. All Configuration Options are always rejected simultaneously.

5.5. Terminate-Request and Terminate-Ack

Description

LCP includes Terminate-Request and Terminate-Ack Codes in order to provide a mechanism for closing a connection.

An implementation wishing to close a connection SHOULD transmit a Terminate-Request. Terminate-Request packets SHOULD continue to be sent until Terminate-Ack is received, the lower layer indicates that it has gone down, or a sufficiently large number have been transmitted such that the peer is down with reasonable certainty.

Upon reception of a Terminate-Request, a Terminate-Ack MUST be transmitted.

Reception of an unelicited Terminate-Ack indicates that the peer is in the Closed or Stopped states, or is otherwise in need of re-negotiation.

A summary of the Terminate-Request and Terminate-Ack packet formats is shown below. The fields are transmitted from left to right.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Data ...
+-+-+-+-+
```

Code

5 for Terminate-Request;

6 for Terminate-Ack.

Identifier

On transmission, the Identifier field MUST be changed whenever the content of the Data field changes, and whenever a valid reply has been received for a previous request. For retransmissions, the Identifier MAY remain unchanged.

On reception, the Identifier field of the Terminate-Request is copied into the Identifier field of the Terminate-Ack packet.

Data

The Data field is zero or more octets, and contains uninterpreted data for use by the sender. The data may consist of any binary value. The end of the field is indicated by the Length.

5.6. Code-Reject

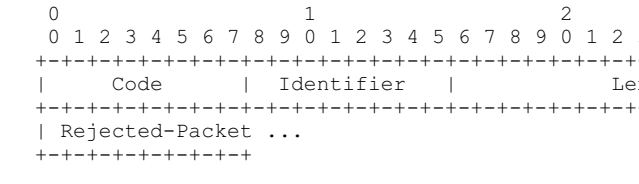
Description

Reception of a LCP packet with an unknown Code indicates that the peer is operating with a different version. This MUST be reported

back to the sender of the unknown Code by transmitting a Code-Reject.

Upon reception of the Code-Reject of a code which is fundamental to this version of the protocol, the implementation SHOULD report the problem and drop the connection, since it is unlikely that the situation can be rectified automatically.

A summary of the Code-Reject packet format is shown below. The fields are transmitted from left to right.



Code

7 for Code-Reject.

Identifier

The Identifier field MUST be changed for each Code-Reject sent.

Rejected-Packet

The Rejected-Packet field contains a copy of the LCP packet which is being rejected. It begins with the Information field, and does not include any Data Link Layer headers nor an FCS. The Rejected-Packet MUST be truncated to comply with the peer's

established MRU.

5.7. Protocol-Reject

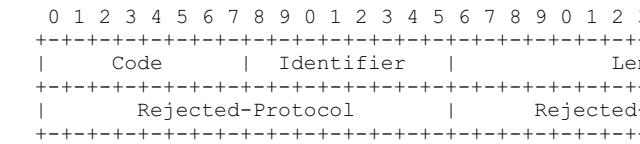
Description

Reception of a PPP packet with an unknown Protocol field indicates that the peer is attempting to use a protocol which is unsupported. This usually occurs when the peer attempts to configure a new protocol. If the LCP automaton is in the Opened state, then this MUST be reported back to the peer by transmitting a Protocol-Reject.

Upon reception of a Protocol-Reject, the implementation MUST stop sending packets of the indicated protocol at the earliest opportunity.

Protocol-Reject packets can only be sent in the LCP Opened state. Protocol-Reject packets received in any state other than the LCP Opened state SHOULD be silently discarded.

A summary of the Protocol-Reject packet format is shown below. The fields are transmitted from left to right.



Code

8 for Protocol-Reject.

Identifier

The Identifier field MUST be changed for each Protocol-Reject sent.

Rejected-Protocol

The Rejected-Protocol field is two octets, and contains the PPP Protocol field of the packet which is being rejected.

Rejected-Information

The Rejected-Information field contains a copy of the packet which is being rejected. It begins with the Information field, and does not include any Data Link Layer headers nor an FCS. The Rejected-Information MUST be truncated to comply with the peer's established MRU.

5.8. Echo-Request and Echo-Reply

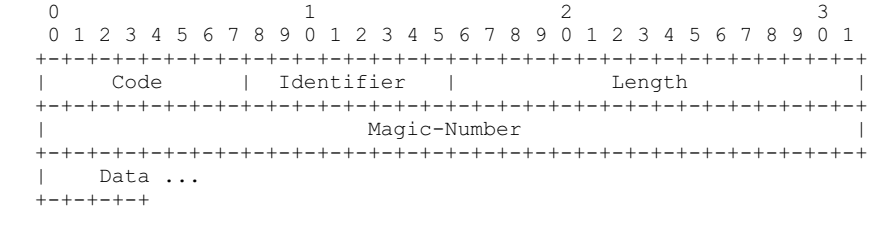
Description

LCP includes Echo-Request and Echo-Reply Codes in order to provide a Data Link Layer loopback mechanism for use in exercising both directions of the link. This is useful as an aid in debugging, link quality determination, performance testing, and for numerous other functions.

Upon reception of an Echo-Request in the LCP Opened state, an Echo-Reply MUST be transmitted.

Echo-Request and Echo-Reply packets MUST only be sent in the LCP Opened state. Echo-Request and Echo-Reply packets received in any state other than the LCP Opened state SHOULD be silently discarded.

A summary of the Echo-Request and Echo-Reply packet formats is shown below. The fields are transmitted from left to right.



Code

9 for Echo-Request;

10 for Echo-Reply.

Identifier

On transmission, the Identifier field MUST be changed whenever the content of the Data field changes, and whenever a valid reply has been received for a previous request. For retransmissions, the Identifier MAY remain unchanged.

On reception, the Identifier field of the Echo-Request is copied into the Identifier field of the Echo-Reply packet.

Magic-Number

The Magic-Number field is four octets, and aids in detecting links which are in the looped-back condition. Until the Magic-Number Configuration Option has been successfully negotiated, the Magic-Number MUST be transmitted as zero. See the Magic-Number Configuration Option for further explanation.

Data

The Data field is zero or more octets, and contains uninterpreted data for use by the sender. The data may consist of any binary value. The end of the field is indicated by the Length.

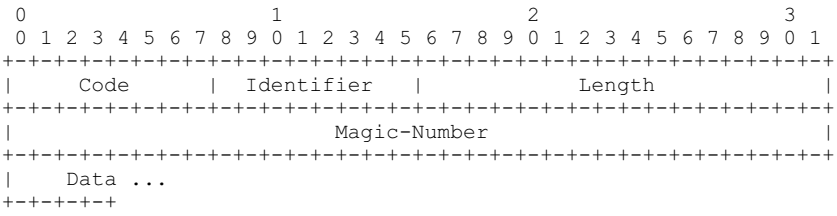
5.9. Discard-Request

Description

LCP includes a Discard-Request Code in order to provide a Data Link Layer sink mechanism for use in exercising the local to remote direction of the link. This is useful as an aid in debugging, performance testing, and for numerous other functions.

Discard-Request packets MUST only be sent in the LCP Opened state. On reception, the receiver MUST silently discard any Discard-Request that it receives.

A summary of the Discard-Request packet format is shown below. The fields are transmitted from left to right.



Code

11 for Discard-Request.

Identifier

The Identifier field MUST be changed for each Discard-Request sent.

Magic-Number

The Magic-Number field is four octets, and aids in detecting links which are in the looped-back condition. Until the Magic-Number Configuration Option has been successfully negotiated, the Magic-Number MUST be transmitted as zero. See the Magic-Number Configuration Option for further explanation.

Data

The Data field is zero or more octets, and contains uninterpreted data for use by the sender. The data may consist of any binary value. The end of the field is indicated by the Length.

6. LCP Configuration Options

LCP Configuration Options allow negotiation of modifications to the default characteristics of a point-to-point link. If a Configuration Option is not included in a Configure-Request packet, the default value for that Configuration Option is assumed.

Some Configuration Options MAY be listed more than once. The effect of this is Configuration Option specific, and is specified by each such Configuration Option description. (None of the Configuration Options in this specification can be listed more than once.)

The end of the list of Configuration Options is indicated by the Length field of the LCP packet.

Unless otherwise specified, all Configuration Options apply in a half-duplex fashion; typically, in the receive direction of the link from the point of view of the Configure-Request sender.

Design Philosophy

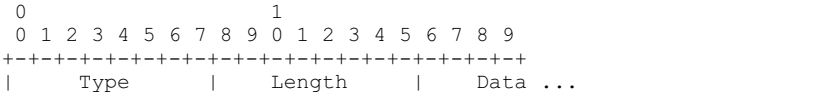
The options indicate additional capabilities or requirements of the implementation that is requesting the option. An implementation which does not understand any option SHOULD interoperate with one which implements every option.

A default is specified for each option which allows the link to correctly function without negotiation of the option, although perhaps with less than optimal performance.

Except where explicitly specified, acknowledgement of an option does not require the peer to take any additional action other than the default.

It is not necessary to send the default values for the options in a Configure-Request.

A summary of the Configuration Option format is shown below. The fields are transmitted from left to right.



```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

The Type field is one octet, and indicates the type of Configuration Option. Up-to-date values of the LCP Option Type field are specified in the most recent "Assigned Numbers" RFC [2]. This document concerns the following values:

```
0       RESERVED
1       Maximum-Receive-Unit
3       Authentication-Protocol
4       Quality-Protocol
5       Magic-Number
7       Protocol-Field-Compression
8       Address-and-Control-Field-Compression
```

Length

The Length field is one octet, and indicates the length of this Configuration Option including the Type, Length and Data fields.

If a negotiable Configuration Option is received in a Configure-Request, but with an invalid or unrecognized Length, a Configure-Nak SHOULD be transmitted which includes the desired Configuration Option with an appropriate Length and Data.

Data

The Data field is zero or more octets, and contains information specific to the Configuration Option. The format and length of the Data field is determined by the Type and Length fields.

When the Data field is indicated by the Length to extend beyond the end of the Information field, the entire packet is silently discarded without affecting the automaton.

6.1. Maximum-Receive-Unit (MRU)

Description

This Configuration Option may be sent to inform the peer that the implementation can receive larger packets, or to request that the peer send smaller packets.

The default value is 1500 octets. If smaller packets are requested, an implementation MUST still be able to receive the full 1500 octet information field in case link synchronization is lost.

Implementation Note:

This option is used to indicate an implementation capability. The peer is not required to maximize the use of the capacity. For example, when a MRU is indicated which is 2048 octets, the peer is not required to send any packet with 2048 octets. The peer need not Configure-Nak to indicate that it will only send smaller packets, since the implementation will always require support for at least 1500 octets.

A summary of the Maximum-Receive-Unit Configuration Option format is shown below. The fields are transmitted from left to right.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     |      Maximum-Receive-Unit     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

1

Length

4

Maximum-Receive-Unit

The Maximum-Receive-Unit field is two octets, and specifies the maximum number of octets in the Information and Padding fields.

It does not include the framing, Protocol field, FCS, nor any transparency bits or bytes.

6.2. Authentication-Protocol

Description

On some links it may be desirable to require a peer to authenticate itself before allowing network-layer protocol packets to be exchanged.

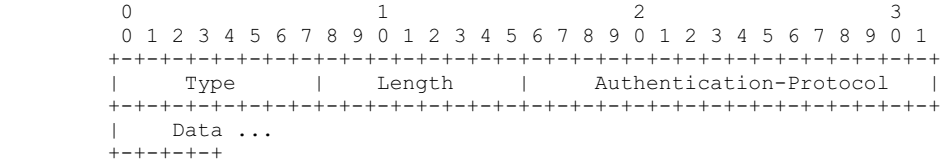
This Configuration Option provides a method to negotiate the use of a specific protocol for authentication. By default, authentication is not required.

An implementation MUST NOT include multiple Authentication-Protocol Configuration Options in its Configure-Request packets. Instead, it SHOULD attempt to configure the most desirable protocol first. If that protocol is Configure-Nak'd, then the implementation SHOULD attempt the next most desirable protocol in the next Configure-Request.

The implementation sending the Configure-Request is indicating that it expects authentication from its peer. If an implementation sends a Configure-Ack, then it is agreeing to authenticate with the specified protocol. An implementation receiving a Configure-Ack SHOULD expect the peer to authenticate with the acknowledged protocol.

There is no requirement that authentication be full-duplex or that the same protocol be used in both directions. It is perfectly acceptable for different protocols to be used in each direction. This will, of course, depend on the specific protocols negotiated.

A summary of the Authentication-Protocol Configuration Option format is shown below. The fields are transmitted from left to right.



Type

3

Length

>= 4

Authentication-Protocol

The Authentication-Protocol field is two octets, and indicates the authentication protocol desired. Values for this field are always the same as the PPP Protocol field values for that same authentication protocol.

Up-to-date values of the Authentication-Protocol field are specified in the most recent "Assigned Numbers" RFC [2]. Current values are assigned as follows:

Value (in hex)	Protocol
c023	Password Authentication Protocol
c223	Challenge Handshake Authentication Protocol

Data

The Data field is zero or more octets, and contains additional data as determined by the particular protocol.

6.3. Quality-Protocol

Description

On some links it may be desirable to determine when, and how often, the link is dropping data. This process is called link quality monitoring.

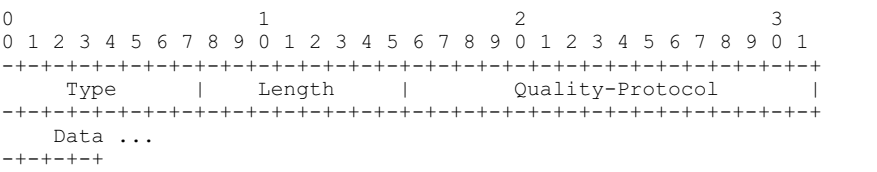
This Configuration Option provides a method to negotiate the use of a specific protocol for link quality monitoring. By default, link quality monitoring is disabled.

The implementation sending the Configure-Request is indicating that it expects to receive monitoring information from its peer. If an implementation sends a Configure-Ack, then it is agreeing to send the specified protocol. An implementation receiving a Configure-Ack SHOULD expect the peer to send the acknowledged protocol.

There is no requirement that quality monitoring be full-duplex or

that the same protocol be used in both directions. It is perfectly acceptable for different protocols to be used in each direction. This will, of course, depend on the specific protocols negotiated.

A summary of the Quality-Protocol Configuration Option format is shown below. The fields are transmitted from left to right.



Type

4

Length

>= 4

Quality-Protocol

The Quality-Protocol field is two octets, and indicates the link quality monitoring protocol desired. Values for this field are always the same as the PPP Protocol field values for that same monitoring protocol.

Up-to-date values of the Quality-Protocol field are specified in the most recent "Assigned Numbers" RFC [2]. Current values are assigned as follows:

Value (in hex)	Protocol
c025	Link Quality Report

Data

The Data field is zero or more octets, and contains additional data as determined by the particular protocol.

6.4. Magic-Number

Description

This Configuration Option provides a method to detect looped-back links and other Data Link Layer anomalies. This Configuration Option MAY be required by some other Configuration Options such as the Quality-Protocol Configuration Option. By default, the Magic-Number is not negotiated, and zero is inserted where a Magic-Number might otherwise be used.

Before this Configuration Option is requested, an implementation MUST choose its Magic-Number. It is recommended that the Magic-Number be chosen in the most random manner possible in order to guarantee with very high probability that an implementation will arrive at a unique number. A good way to choose a unique random number is to start with a unique seed. Suggested sources of uniqueness include machine serial numbers, other network hardware addresses, time-of-day clocks, etc. Particularly good random number seeds are precise measurements of the inter-arrival time of physical events such as packet reception on other connected networks, server response time, or the typing rate of a human user. It is also suggested that as many sources as possible be

used simultaneously.

When a Configure-Request is received with a Magic-Number Configuration Option, the received Magic-Number is compared with the Magic-Number of the last Configure-Request sent to the peer. If the two Magic-Numbers are different, then the link is not looped-back, and the Magic-Number SHOULD be acknowledged. If the two Magic-Numbers are equal, then it is possible, but not certain, that the link is looped-back and that this Configure-Request is actually the one last sent. To determine this, a Configure-Nak MUST be sent specifying a different Magic-Number value. A new Configure-Request SHOULD NOT be sent to the peer until normal processing would cause it to be sent (that is, until a Configure-Nak is received or the Restart timer runs out).

Reception of a Configure-Nak with a Magic-Number different from that of the last Configure-Nak sent to the peer proves that a link is not looped-back, and indicates a unique Magic-Number. If the Magic-Number is equal to the one sent in the last Configure-Nak, the possibility of a looped-back link is increased, and a new Magic-Number MUST be chosen. In either case, a new Configure-Request SHOULD be sent with the new Magic-Number.

If the link is indeed looped-back, this sequence (transmit Configure-Request, receive Configure-Request, transmit Configure-

Nak, receive Configure-Nak) will repeat over and over again. If the link is not looped-back, this sequence might occur a few times, but it is extremely unlikely to occur repeatedly. More likely, the Magic-Numbers chosen at either end will quickly diverge, terminating the sequence. The following table shows the probability of collisions assuming that both ends of the link select Magic-Numbers with a perfectly uniform distribution:

Number of Collisions	Probability
1	1/2**32 = 2.3 E-10
2	1/2**32**2 = 5.4 E-20
3	1/2**32**3 = 1.3 E-29

Good sources of uniqueness or randomness are required for this divergence to occur. If a good source of uniqueness cannot be found, it is recommended that this Configuration Option not be enabled; Configure-Requests with the option SHOULD NOT be transmitted and any Magic-Number Configuration Options which the peer sends SHOULD be either acknowledged or rejected. In this case, looped-back links cannot be reliably detected by the implementation, although they may still be detectable by the peer.

If an implementation does transmit a Configure-Request with a Magic-Number Configuration Option, then it MUST NOT respond with a Configure-Reject when it receives a Configure-Request with a Magic-Number Configuration Option. That is, if an implementation desires to use Magic Numbers, then it MUST also allow its peer to do so. If an implementation does receive a Configure-Reject in response to a Configure-Request, it can only mean that the link is not looped-back, and that its peer will not be using Magic-Numbers. In this case, an implementation SHOULD act as if the negotiation had been successful (as if it had instead received a Configure-Ack).

The Magic-Number also may be used to detect looped-back links during normal operation, as well as during Configuration Option negotiation. All LCP Echo-Request, Echo-Reply, and Discard-Request packets have a Magic-Number field. If Magic-Number has been successfully negotiated, an implementation MUST transmit these packets with the Magic-Number field set to its negotiated Magic-Number.

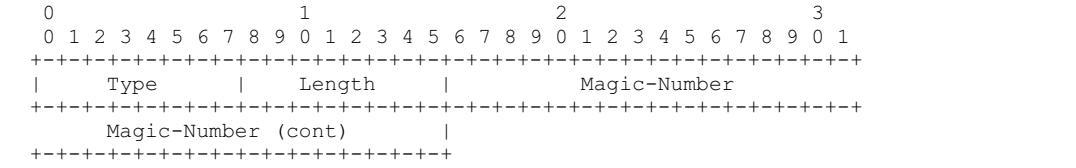
The Magic-Number field of these packets SHOULD be inspected on reception. All received Magic-Number fields MUST be equal to either zero or the peer's unique Magic-Number, depending on whether or not the peer negotiated a Magic-Number.

Reception of a Magic-Number field equal to the negotiated local Magic-Number indicates a looped-back link. Reception of a Magic-Number other than the negotiated local Magic-Number, the peer's negotiated Magic-Number, or zero if the peer didn't negotiate one, indicates a link which has been (mis)configured for communications

with a different peer.

Procedures for recovery from either case are unspecified, and may vary from implementation to implementation. A somewhat pessimistic procedure is to assume a LCP Down event. A further Open event will begin the process of re-establishing the link, which can't complete until the looped-back condition is terminated, and Magic-Numbers are successfully negotiated. A more optimistic procedure (in the case of a looped-back link) is to begin transmitting LCP Echo-Request packets until an appropriate Echo-Reply is received, indicating a termination of the looped-back condition.

A summary of the Magic-Number Configuration Option format is shown below. The fields are transmitted from left to right.



Type

5

Length

6

Magic-Number

The Magic-Number field is four octets, and indicates a number which is very likely to be unique to one end of the link. A Magic-Number of zero is illegal and MUST always be Nak'd, if it is not Rejected outright.

6.5. Protocol-Field-Compression (PFC)

Description

This Configuration Option provides a method to negotiate the compression of the PPP Protocol field. By default, all implementations MUST transmit packets with two octet PPP Protocol fields.

PPP Protocol field numbers are chosen such that some values may be compressed into a single octet form which is clearly distinguishable from the two octet form. This Configuration Option is sent to inform the peer that the implementation can receive such single octet Protocol fields.

As previously mentioned, the Protocol field uses an extension mechanism consistent with the ISO 3309 extension mechanism for the Address field; the Least Significant Bit (LSB) of each octet is used to indicate extension of the Protocol field. A binary "0" as the LSB indicates that the Protocol field continues with the following octet. The presence of a binary "1" as the LSB marks the last octet of the Protocol field. Notice that any number of "0" octets may be prepended to the field, and will still indicate the same value (consider the two binary representations for 3, 00000011 and 00000000 00000011).

When using low speed links, it is desirable to conserve bandwidth by sending as little redundant data as possible. The Protocol-Field-Compression Configuration Option allows a trade-off between implementation simplicity and bandwidth efficiency. If successfully negotiated, the ISO 3309 extension mechanism may be used to compress the Protocol field to one octet instead of two. The large majority of packets are compressible since data protocols are typically assigned with Protocol field values less than 256.

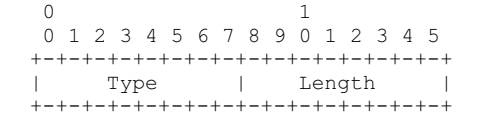
Compressed Protocol fields MUST NOT be transmitted unless this Configuration Option has been negotiated. When negotiated, PPP implementations MUST accept PPP packets with either double-octet or single-octet Protocol fields, and MUST NOT distinguish between them.

The Protocol field is never compressed when sending any LCP packet. This rule guarantees unambiguous recognition of LCP packets.

When a Protocol field is compressed, the Data Link Layer FCS field is calculated on the compressed frame, not the original

uncompressed frame.

A summary of the Protocol-Field-Compression Configuration Option format is shown below. The fields are transmitted from left to right.



Type

7

Length

2

6.6. Address-and-Control-Field-Compression (ACFC)

Description

This Configuration Option provides a method to negotiate the compression of the Data Link Layer Address and Control fields. By default, all implementations MUST transmit frames with Address and Control fields appropriate to the link framing.

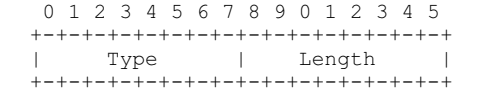
Since these fields usually have constant values for point-to-point links, they are easily compressed. This Configuration Option is sent to inform the peer that the implementation can receive compressed Address and Control fields.

If a compressed frame is received when Address-and-Control-Field-Compression has not been negotiated, the implementation MAY silently discard the frame.

The Address and Control fields MUST NOT be compressed when sending any LCP packet. This rule guarantees unambiguous recognition of LCP packets.

When the Address and Control fields are compressed, the Data Link Layer FCS field is calculated on the compressed frame, not the original uncompressed frame.

A summary of the Address-and-Control-Field-Compression configuration option format is shown below. The fields are transmitted from left to right.



Type

8

Length

2

Security Considerations

Security issues are briefly discussed in sections concerning the Authentication Phase, the Close event, and the Authentication-Protocol Configuration Option.

References

[1] Perkins, D., "Requirements for an Internet Standard Point-to-

Point Protocol", RFC 1547, Carnegie Mellon University, December 1993.

[2] Reynolds, J., and Postel, J., "Assigned Numbers", STD 2, RFC 1340, USC/Information Sciences Institute, July 1992.

Acknowledgements

This document is the product of the Point-to-Point Protocol Working Group of the Internet Engineering Task Force (IETF). Comments should be submitted to the ietf-ppp@merit.edu mailing list.

Much of the text in this document is taken from the working group requirements [1]; and RFCs 1171 & 1172, by Drew Perkins while at Carnegie Mellon University, and by Russ Hobby of the University of California at Davis.

William Simpson was principally responsible for introducing consistent terminology and philosophy, and the re-design of the phase and negotiation state machines.

Many people spent significant time helping to develop the Point-to-Point Protocol. The complete list of people is too numerous to list, but the following people deserve special thanks: Rick Adams, Ken Adelman, Fred Baker, Mike Ballard, Craig Fox, Karl Fox, Phill Gross, Kory Hamzeh, former WG chair Russ Hobby, David Kaufman, former WG chair Steve Knowles, Mark Lewis, former WG chair Brian Lloyd, John LoVerso, Bill Melohn, Mike Patton, former WG chair Drew Perkins, Greg Satz, John Shriver, Vernon Schryver, and Asher Waldfogel.

Special thanks to Morning Star Technologies for providing computing resources and network access support for writing this specification.

Chair's Address

The working group can be contacted via the current chair:

Fred Baker
Advanced Computer Communications
315 Bollay Drive
Santa Barbara, California 93117

fbaker@acc.com

Editor's Address

Questions about this memo can also be directed to:

William Allen Simpson
Daydreamer
Computer Systems Consulting Services
1384 Fontaine
Madison Heights, Michigan 48071

Bill.Simpson@um.cc.umich.edu
bsimpson@MorningStar.com

Comment on RFC 1661

Previous: RFC 1660 - Definitions of Managed Objects for Parallel-printer-like Hardware Devices using SMIv2

Next: RFC 1662 - PPP in HDLC-like Framing

[RFC Index | RFC Search | Usenet FAQs | Web FAQs | Documents | Cities]

RFC 791 (RFC791)

Internet RFC/STD/FYI/BCP Archives

RFC 791 - Internet Protocol

RFC: 791

INTERNET PROTOCOL

DARPA INTERNET PROGRAM

PROTOCOL SPECIFICATION

September 1981

prepared for

Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

by

Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

September 1981

TABLE OF CONTENTS

PREFACE

This document specifies the DoD Standard Internet Protocol. This document is based on six earlier editions of the ARPA Internet Protocol Specification, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition revises aspects of addressing, error handling, option codes, and the security, precedence, compartments, and handling restriction features of the internet protocol.

Jon Postel

Editor

September 1981

RFC: 791
Replaces: RFC 760
IENs 128, 123, 111,
80, 54, 44, 41, 28, 26

INTERNET PROTOCOL

DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

1. INTRODUCTION

1.1. Motivation

The Internet Protocol is designed for use in interconnected systems of packet-switched computer communication networks. Such a system has been called a "catenet" [1]. The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The internet protocol also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks.

1.2. Scope

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols. The internet protocol can capitalize on the services of its supporting networks to provide various types and qualities of service.

1.3. Interfaces

This protocol is called on by host-to-host protocols in an internet environment. This protocol calls on local network protocols to carry the internet datagram to the next gateway or destination host.

For example, a TCP module would call on the internet module to take a TCP segment (including the TCP header and user data) as the data portion of an internet datagram. The TCP module would provide the addresses and other parameters in the internet header to the internet module as arguments of the call. The internet module would then create an internet datagram and call on the local network interface to transmit the internet datagram.

In the ARPANET case, for example, the internet module would call on a

local net module which would add the 1822 leader [2] to the internet datagram creating an ARPANET message to transmit to the IMP. The ARPANET address would be derived from the internet address by the local network interface and would be the address of some host in the ARPANET, that host might be a gateway to other networks.

1.4. Operation

The internet protocol implements two basic functions: addressing and fragmentation.

The internet modules use the addresses carried in the internet header to transmit internet datagrams toward their destinations. The selection of a path for transmission is called routing.

The internet modules use fields in the internet header to fragment and reassemble internet datagrams when necessary for transmission through "small packet" networks.

The model of operation is that an internet module resides in each host engaged in internet communication and in each gateway that interconnects networks. These modules share common rules for interpreting address fields and for fragmenting and assembling internet datagrams. In addition, these modules (especially in gateways) have procedures for making routing decisions and other functions.

The internet protocol treats each internet datagram as an independent entity unrelated to any other internet datagram. There are no connections or logical circuits (virtual or otherwise).

The internet protocol uses four key mechanisms in providing its service: Type of Service, Time to Live, Options, and Header Checksum.

The Type of Service is used to indicate the quality of the service desired. The type of service is an abstract or generalized set of parameters which characterize the service choices provided in the networks that make up the internet. This type of service indication is to be used by gateways to select the actual transmission parameters for a particular network, the network to be used for the next hop, or the next gateway when routing an internet datagram.

The Time to Live is an indication of an upper bound on the lifetime of an internet datagram. It is set by the sender of the datagram and reduced at the points along the route where it is processed. If the time to live reaches zero before the internet datagram reaches its destination, the internet datagram is destroyed. The time to live can be thought of as a self destruct time limit.

The Options provide for control functions needed or useful in some situations but unnecessary for the most common communications. The options include provisions for timestamps, security, and special routing.

The Header Checksum provides a verification that the information used in processing internet datagram has been transmitted correctly. The data may contain errors. If the header checksum fails, the internet datagram is discarded at once by the entity which detects the error.

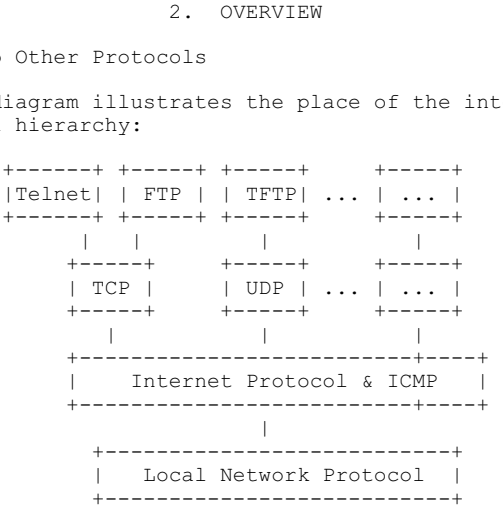
The internet protocol does not provide a reliable communication facility. There are no acknowledgments either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is no flow control.

Errors detected may be reported via the Internet Control Message Protocol (ICMP) [3] which is implemented in the internet protocol module.

2. OVERVIEW

2.1. Relation to Other Protocols

The following diagram illustrates the place of the internet protocol in the protocol hierarchy:



Protocol Relationships

Figure 1.

Internet protocol interfaces on one side to the higher level host-to-host protocols and on the other side to the local network protocol. In this context a "local network" may be a small network in a building or a large network such as the ARPANET.

2.2. Model of Operation

The model of operation for transmitting a datagram from one application program to another is illustrated by the following scenario:

We suppose that this transmission will involve one intermediate gateway.

The sending application program prepares its data and calls on its local internet module to send that data as a datagram and passes the destination address and other parameters as arguments of the call.

The internet module prepares a datagram header and attaches the data to it. The internet module determines a local network address for this internet address, in this case it is the address of a gateway.

It sends this datagram and the local network address to the local network interface.

The local network interface creates a local network header, and attaches the datagram to it, then sends the result via the local network.

The datagram arrives at a gateway host wrapped in the local network header, the local network interface strips off this header, and turns the datagram over to the internet module. The internet module determines from the internet address that the datagram is to be forwarded to another host in a second network. The internet module determines a local net address for the destination host. It calls on the local network interface for that network to send the datagram.

This local network interface creates a local network header and attaches the datagram sending the result to the destination host.

At this destination host the datagram is stripped of the local net header by the local network interface and handed to the internet module.

The internet module determines that the datagram is for an application program in this host. It passes the data to the application program in response to a system call, passing the source address and other parameters as results of the call.

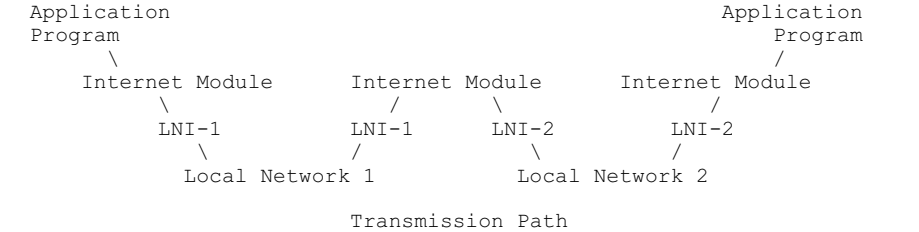


Figure 2

2.3. Function Description

The function or purpose of Internet Protocol is to move datagrams through an interconnected set of networks. This is done by passing the datagrams from one internet module to another until the destination is reached. The internet modules reside in hosts and gateways in the internet system. The datagrams are routed from one internet module to another through individual networks based on the interpretation of an internet address. Thus, one important mechanism of the internet protocol is the internet address.

In the routing of messages from one internet module to another, datagrams may need to traverse a network whose maximum packet size is smaller than the size of the datagram. To overcome this difficulty, a fragmentation mechanism is provided in the internet protocol.

Addressing

A distinction is made between names, addresses, and routes [4]. A name indicates what we seek. An address indicates where it is. A route indicates how to get there. The internet protocol deals primarily with addresses. It is the task of higher level (i.e., host-to-host or application) protocols to make the mapping from names to addresses. The internet module maps internet addresses to local net addresses. It is the task of lower level (i.e., local net or gateways) procedures to make the mapping from local net addresses to routes.

Addresses are fixed length of four octets (32 bits). An address begins with a network number, followed by local address (called the "rest" field). There are three formats or classes of internet addresses: in class a, the high order bit is zero, the next 7 bits are the network, and the last 24 bits are the local address; in class b, the high order two bits are one-zero, the next 14 bits are the network and the last 16 bits are the local address; in class c, the high order three bits are one-one-zero, the next 21 bits are the network and the last 8 bits are the local address.

Care must be taken in mapping internet addresses to local net addresses; a single physical host must be able to act as if it were several distinct hosts to the extent of using several distinct internet addresses. Some hosts will also have several physical interfaces (multi-homing).

That is, provision must be made for a host to have several physical interfaces to the network with each having several logical internet addresses.

Examples of address mappings may be found in "Address Mappings" [5].

Fragmentation

Fragmentation of an internet datagram is necessary when it originates in a local net that allows a large packet size and must traverse a local net that limits packets to a smaller size to reach its destination.

An internet datagram can be marked "don't fragment." Any internet datagram so marked is not to be internet fragmented under any circumstances. If internet datagram marked don't fragment cannot be delivered to its destination without fragmenting it, it is to be discarded instead.

Fragmentation, transmission and reassembly across a local network which is invisible to the internet protocol module is called intranet fragmentation and may be used [6].

The internet fragmentation and reassembly procedure needs to be able to break a datagram into an almost arbitrary number of pieces that can be later reassembled. The receiver of the fragments uses the identification field to ensure that fragments of different datagrams are not mixed. The fragment offset field tells the receiver the position of a fragment in the original datagram. The fragment offset and length determine the portion of the original datagram covered by this fragment. The more-fragments flag indicates (by being reset) the last fragment. These fields provide sufficient information to reassemble datagrams.

The identification field is used to distinguish the fragments of one datagram from those of another. The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram sets the more-fragments flag to zero and the fragment offset to zero.

To fragment a long internet datagram, an internet protocol module (for example, in a gateway), creates two new internet datagrams and copies the contents of the internet header fields from the long datagram into both new internet headers. The data of the long datagram is divided into two portions on a 8 octet (64 bit) boundary (the second portion might not be an integral multiple of 8 octets, but the first must be). Call the number of 8 octet blocks in the first portion NFB (for Number of Fragment Blocks). The first portion of the data is placed in the first new internet datagram, and the total length field is set to the length of the first

datagram. The more-fragments flag is set to one. The second portion of the data is placed in the second new internet datagram, and the total length field is set to the length of the second datagram. The more-fragments flag carries the same value as the long datagram. The fragment offset field of the second new internet datagram is set to the value of that field in the long datagram plus NFB.

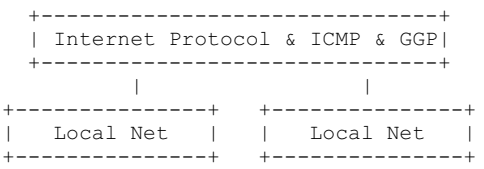
This procedure can be generalized for an n-way split, rather than the two-way split described.

To assemble the fragments of an internet datagram, an internet protocol module (for example at a destination host) combines internet datagrams that all have the same value for the four fields: identification, source, destination, and protocol. The combination is done by placing the data portion of each fragment in the relative position indicated by the fragment offset in that fragment's internet header. The first fragment will have the fragment offset zero, and the last fragment will have the more-fragments flag reset to zero.

2.4. Gateways

Gateways implement internet protocol to forward datagrams between networks. Gateways also implement the Gateway to Gateway Protocol (GGP) [7] to coordinate routing and other internet control information.

In a gateway the higher level protocols need not be implemented and the GGP functions are added to the IP module.



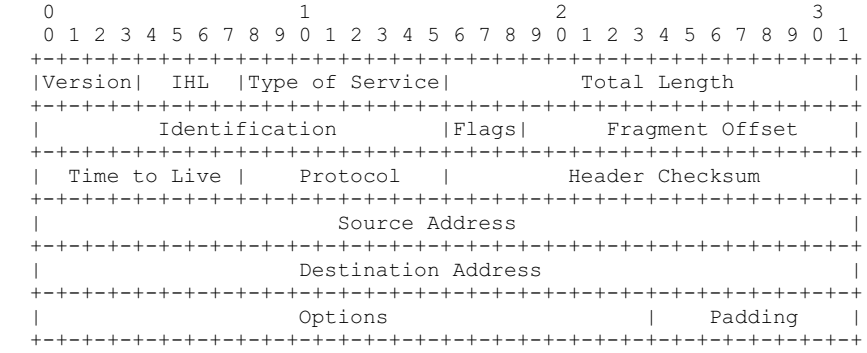
Gateway Protocols

Figure 3.

3. SPECIFICATION

3.1. Internet Header Format

A summary of the contents of the internet header follows:



Example Internet Datagram Header

Figure 4.

Note that each tick mark represents one bit position.

Version: 4 bits

The Version field indicates the format of the internet header. This document describes version 4.

IHL: 4 bits

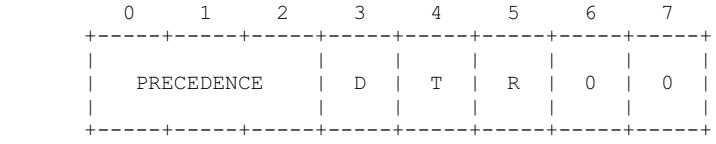
Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.

Type of Service: 8 bits

The Type of Service provides an indication of the abstract

parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic (generally by accepting only traffic above a certain precedence at time of high load). The major choice is a three way tradeoff between low-delay, high-reliability, and high-throughput.

```
  Bits 0-2:  Precedence.
  Bit    3:  0 = Normal Delay,      1 = Low Delay.
  Bits   4:  0 = Normal Throughput, 1 = High Throughput.
  Bits   5:  0 = Normal Relibility, 1 = High Relibility.
  Bit  6-7:  Reserved for Future Use.
```



Precedence

111 - Network Control
110 - Internetwork Control
101 - CRITIC/ECP
100 - Flash Override
011 - Flash
010 - Immediate
001 - Priority
000 - Routine

The use of the Delay, Throughput, and Reliability indications may increase the cost (in some sense) of the service. In many networks better performance for one of these parameters is coupled with worse performance on another. Except for very unusual cases at most two of these three indications should be set.

The type of service is used to specify the treatment of the datagram during its transmission through the internet system. Example mappings of the internet type of service to the actual service provided on networks such as AUTODIN II, ARPANET, SATNET, and PRNET is given in "Service Mappings" [8].

The Network Control precedence designation is intended to be used within a network only. The actual use and control of that designation is up to each network. The Internetwork Control designation is intended for use by gateway control originators only. If the actual use of these precedence designations is of concern to a particular network, it is the responsibility of that network to control the access to, and use of, those precedence designations.

Total Length: 16 bits

Total Length is the length of the datagram, measured in octets, including internet header and data. This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments). It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams.

The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximal internet header is 60 octets, and a typical internet header is 20 octets, allowing a margin for headers of higher level protocols.

Identification: 16 bits

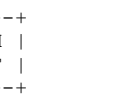
An identifying value assigned by the sender to aid in assembling the

fragments of a datagram.

Flags: 3 bits

Various Control Flags.

Bit 0: reserved, must be zero
Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.
Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.



Fragment Offset: 13 bits

This field indicates where in the datagram this fragment belongs.

The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

Time to Live: 8 bits

This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in internet header processing. The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Protocol: 8 bits

This field indicates the next level protocol used in the data portion of the internet datagram. The values for various protocols are specified in "Assigned Numbers" [9].

Header Checksum: 16 bits

A checksum on the header only. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed.

The checksum algorithm is:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

Source Address: 32 bits

The source address. See section 3.2.

Destination Address: 32 bits

The destination address. See section 3.2.

Options: variable

The options may appear or not in datagrams. They must be

implemented by all IP modules (host and gateways). What is optional is their transmission in any particular datagram, not their implementation.

In some environments the security option may be required in all datagrams.

The option field is variable in length. There may be zero or more options. There are two cases for the format of an option:

Case 1: A single octet of option-type.

Case 2: An option-type octet, an option-length octet, and the actual option-data octets.

The option-length octet counts the option-type octet and the option-length octet as well as the option-data octets.

The option-type octet is viewed as having 3 fields:

1 bit copied flag,
2 bits option class,
5 bits option number.

The copied flag indicates that this option is copied into all fragments on fragmentation.

0 = not copied
1 = copied

The option classes are:

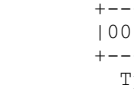
0 = control
1 = reserved for future use
2 = debugging and measurement
3 = reserved for future use

The following internet options are defined:

CLASS	NUMBER	LENGTH	DESCRIPTION
0	0	-	End of Option list. This option occupies only 1 octet; it has no length octet.
0	1	-	No Operation. This option occupies only 1 octet; it has no length octet.
0	2	11	Security. Used to carry Security, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with DOD requirements.
0	3	var.	Loose Source Routing. Used to route the internet datagram based on information supplied by the source.
0	9	var.	Strict Source Routing. Used to route the internet datagram based on information supplied by the source.
0	7	var.	Record Route. Used to trace the route an internet datagram takes.
0	8	4	Stream ID. Used to carry the stream identifier.
2	4	var.	Internet Timestamp.

Specific Option Definitions

End of Option List



Type=0

This option indicates the end of the option list. This might not coincide with the end of the internet header according to the internet header length. This is used at the end of all options, not the end of each option, and need only be used if

the end of the options would not otherwise coincide with the end of the internet header.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

No Operation

```
+--------+
|00000001|
+--------+
  Type=1
```

This option may be used between options, for example, to align the beginning of a subsequent option on a 32 bit boundary.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

Security

This option provides a way for hosts to send security, compartmentation, handling restrictions, and TCC (closed user group) parameters. The format for this option is as follows:

```
+--------+--------+---//---+---//---+---//---+---//---+
|10000010|00001011|SSS  SSS|CCC  CCC|HHH  HHH|  TCC   |
+--------+--------+---//---+---//---+---//---+---//---+
 Type=130 Length=11
```

Security (S field): 16 bits

Specifies one of 16 levels of security (eight of which are reserved for future use).

```
00000000 00000000 - Unclassified
11110001 00110101 - Confidential
01111000 10011010 - EFTO
10111100 01001101 - MMMM
01011110 00100110 - PROG
10101111 00010011 - Restricted
11010111 10001000 - Secret
01101011 11000101 - Top Secret
00110101 11100010 - (Reserved for future use)
10011010 11110001 - (Reserved for future use)
01001101 01111000 - (Reserved for future use)
00100100 10111101 - (Reserved for future use)
00010011 01011110 - (Reserved for future use)
10001001 10101111 - (Reserved for future use)
11000100 11010110 - (Reserved for future use)
11100010 01101011 - (Reserved for future use)
```


Compartments (C field): 16 bits

An all zero value is used when the information transmitted is not compartmented. Other values for the compartments field may be obtained from the Defense Intelligence Agency.

Handling Restrictions (H field): 16 bits

The values for the control and release markings are alphanumeric digraphs and are defined in the Defense Intelligence Agency Manual DIAM 65-19, "Standard Security Markings".

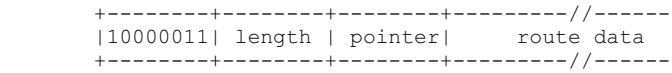
Transmission Control Code (TCC field): 24 bits

Provides a means to segregate traffic and define controlled

communities of interest among subscribers. The TCC values are trigraphs, and are available from HQ DCA Code 530.

Must be copied on fragmentation. This option appears at most once in a datagram.

Loose Source and Record Route



The loose source and record route (LSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

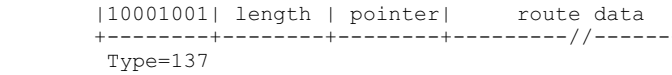
The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a loose source route because the gateway or host IP is allowed to use any route of any number of other intermediate gateways to reach the next address in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

Strict Source and Record Route



The strict source and record route (SSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be

processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the

recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

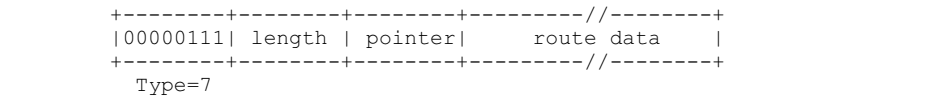
The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a strict source route because the gateway or host IP must send the datagram directly to the next address in the source route through only the directly connected network indicated in the next address to reach the next gateway or host specified in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

Record Route



The record route option provides a means to record the route of an internet datagram.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next area to store a route address. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A recorded route is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is

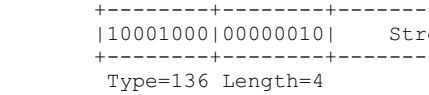
greater than the length, the recorded route data area is full. The originating host must compose this option with a large enough route data area to hold all the address expected. The size of the option does not change due to adding addresses. The intitial contents of the route data area must be zero.

When an internet module routes a datagram it checks to see if the record route option is present. If it is, it inserts its own internet address as known in the environment into which this datagram is being forwarded into the recorded route begining at the octet indicated by the pointer, and increments the pointer by four.

If the route data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the address into the recorded route. If there is some room but not enough room for a full address to be inserted, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host [3].

Not copied on fragmentation, goes in first fragment only. Appears at most once in a datagram.

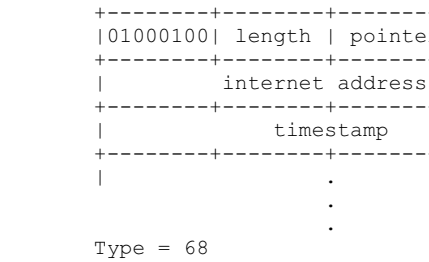
Stream Identifier



This option provides a way for the 16-bit SATNET stream identifier to be carried through networks that do not support the stream concept.

Must be copied on fragmentation. Appears at most once in a datagram.

Internet Timestamp



Type = 68

The Option Length is the number of octets in the option counting the type, length, pointer, and overflow/flag octets (maximum length 40).

The Pointer is the number of octets from the beginning of this option to the end of timestamps plus one (i.e., it points to the octet beginning the space for next timestamp). The smallest legal value is 5. The timestamp area is full when the pointer is greater than the length.

The Overflow (oflw) [4 bits] is the number of IP modules that cannot register timestamps due to lack of space.

The Flag (flg) [4 bits] values are

0 -- time stamps only, stored in consecutive 32-bit words,

1 -- each timestamp is preceded with internet address of the registering entity,

3 -- the internet address fields are prespecified. An IP module only registers its timestamp if it matches its own address with the next specified internet address.

The Timestamp is a right-justified, 32-bit timestamp in milliseconds since midnight UT. If the time is not available in milliseconds or cannot be provided with respect to midnight UT then any time may be inserted as a timestamp provided the high order bit of the timestamp field is set to one to indicate the use of a non-standard value.

The originating host must compose this option with a large enough timestamp data area to hold all the timestamp information expected. The size of the option does not change due to adding

timestamps. The intitial contents of the timestamp data area must be zero or internet address/zero pairs.

If the timestamp data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the timestamp, but the overflow count is incremented by one.

If there is some room but not enough room for a full timestamp to be inserted, or the overflow count itself overflows, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host [3].

The timestamp option is not copied upon fragmentation. It is carried in the first fragment. Appears at most once in a datagram.

Padding: variable

The internet header padding is used to ensure that the internet header ends on a 32 bit boundary. The padding is zero.

3.2. Discussion

The implementation of a protocol must be robust. Each implementation must expect to interoperate with others created by different individuals. While the goal of this specification is to be explicit about the protocol there is the possibility of differing interpretations. In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior. That is, it must be careful to send well-formed datagrams, but must accept any datagram that it can interpret (e.g., not object to technical errors where the meaning is still clear).

The basic internet service is datagram oriented and provides for the fragmentation of datagrams at gateways, with reassembly taking place at the destination internet protocol module in the destination host. Of course, fragmentation and reassembly of datagrams within a network or by private agreement between the gateways of a network is also allowed since this is transparent to the internet protocols and the higher-level protocols. This transparent type of fragmentation and reassembly is termed "network-dependent" (or intranet) fragmentation and is not discussed further here.

Internet addresses distinguish sources and destinations to the host level and provide a protocol field as well. It is assumed that each protocol will provide for whatever multiplexing is necessary within a host.

Addressing

To provide for flexibility in assigning address to networks and allow for the large number of small to intermediate sized networks the interpretation of the address field is coded to specify a small number of networks with a large number of host, a moderate number of networks with a moderate number of hosts, and a large number of networks with a small number of hosts. In addition there is an escape code for extended addressing mode.

Address Formats:

High Order Bits	Format	Class
0	7 bits of net, 24 bits of host	a
10	14 bits of net, 16 bits of host	b
110	21 bits of net, 8 bits of host	c
111	escape to extended addressing mode	

A value of zero in the network field means this network. This is only used in certain ICMP messages. The extended addressing mode is undefined. Both of these features are reserved for future use.

The actual values assigned for network addresses is given in "Assigned Numbers" [9].

The local address, assigned by the local network, must allow for a single physical host to act as several distinct internet hosts. That is, there must be a mapping between internet host addresses and network/host interfaces that allows several internet addresses to correspond to one interface. It must also be allowed for a host to have several physical interfaces and to treat the datagrams from several of them as if they were all addressed to a single host.

Address mappings between internet addresses and addresses for ARPANET, SATNET, PRNET, and other networks are described in "Address Mappings" [5].

Fragmentation and Reassembly.

The internet identification field (ID) is used together with the source and destination address, and the protocol fields, to identify datagram fragments for reassembly.

The More Fragments flag bit (MF) is set if the datagram is not the last fragment. The Fragment Offset field identifies the fragment location, relative to the beginning of the original unfragmented datagram. Fragments are counted in units of 8 octets. The

fragmentation strategy is designed so than an unfragmented datagram has all zero fragmentation information (MF = 0, fragment offset = 0). If an internet datagram is fragmented, its data portion must be broken on 8 octet boundaries.

This format allows 2**13 = 8192 fragments of 8 octets each for a total of 65,536 octets. Note that this is consistent with the the datagram total length field (of course, the header is counted in the total length and not in the fragments).

When fragmentation occurs, some options are copied, but others remain with the first fragment only.

Every internet module must be able to forward a datagram of 68 octets without further fragmentation. This is because an internet header may be up to 60 octets, and the minimum fragment is 8 octets.

Every internet destination must be able to receive a datagram of 576 octets either in one piece or in fragments to be reassembled.

The fields which may be affected by fragmentation include:

(1) options field
(2) more fragments flag
(3) fragment offset
(4) internet header length field
(5) total length field
(6) header checksum

If the Don't Fragment flag (DF) bit is set, then internet fragmentation of this datagram is NOT permitted, although it may be discarded. This can be used to prohibit fragmentation in cases where the receiving host does not have sufficient resources to reassemble internet fragments.

One example of use of the Don't Fragment feature is to down line load a small host. A small host could have a boot strap program that accepts a datagram stores it in memory and then executes it.

The fragmentation and reassembly procedures are most easily described by examples. The following procedures are example implementations.

General notation in the following pseudo programs: "=<" means "less

than or equal", "#" means "not equal", "=" means "equal", "<-" means "is set to". Also, "x to y" includes x and excludes y; for example, "4 to 7" would include 4, 5, and 6 (but not 7).

An Example Fragmentation Procedure

The maximum sized datagram that can be transmitted through the next network is called the maximum transmission unit (MTU).

If the total length is less than or equal the maximum transmission unit then submit this datagram to the next step in datagram processing; otherwise cut the datagram into two fragments, the first fragment being the maximum size, and the second fragment being the rest of the datagram. The first fragment is submitted to the next step in datagram processing, while the second fragment is submitted to this procedure in case it is still too large.

Notation:

```
FO    -  Fragment Offset
IHL   -  Internet Header Length
DF    -  Don't Fragment flag
MF    -  More Fragments flag
TL    -  Total Length
OFO   -  Old Fragment Offset
OIHL  -  Old Internet Header Length
OMF   -  Old More Fragments flag
OTL   -  Old Total Length
NFB   -  Number of Fragment Blocks
MTU   -  Maximum Transmission Unit
```

Procedure:

```
IF TL =< MTU THEN Submit this datagram to the next step
     in datagram processing ELSE IF DF = 1 THEN discard the
datagram ELSE
To produce the first fragment:
(1)  Copy the original internet header;
(2)  OIHL <- IHL; OTL <- TL; OFO <- FO; OMF <- MF;
(3)  NFB <- (MTU-IHL*4)/8;
(4)  Attach the first NFB*8 data octets;
(5)  Correct the header:
     MF <- 1;  TL <- (IHL*4)+(NFB*8);
     Recompute Checksum;
(6)  Submit this fragment to the next step in
     datagram processing;
To produce the second fragment:
(7)  Selectively copy the internet header (some options
     are not copied, see option definitions);
(8)  Append the remaining data;
(9)  Correct the header:
     IHL <- (((OIHL*4)-(length of options not copied))+3)/4;
```



```
     TL <- OTL - NFB*8 - (OIHL-IHL)*4);
     FO <- OFO + NFB;  MF <- OMF;  Recompute Checksum;
(10) Submit this fragment to the fragmentation test; DONE.
```

In the above procedure each fragment (except the last) was made the maximum allowable size. An alternative might produce less than the maximum size datagrams. For example, one could implement a fragmentation procedure that repeatly divided large datagrams in half until the resulting fragments were less than the maximum transmission unit size.

An Example Reassembly Procedure

For each datagram the buffer identifier is computed as the concatenation of the source, destination, protocol, and

identification fields. If this is a whole datagram (that is both the fragment offset and the more fragments fields are zero), then any reassembly resources associated with this buffer identifier are released and the datagram is forwarded to the next step in datagram processing.

If no other fragment with this buffer identifier is on hand then reassembly resources are allocated. The reassembly resources consist of a data buffer, a header buffer, a fragment block bit table, a total data length field, and a timer. The data from the fragment is placed in the data buffer according to its fragment offset and length, and bits are set in the fragment block bit table corresponding to the fragment blocks received.

If this is the first fragment (that is the fragment offset is zero) this header is placed in the header buffer. If this is the last fragment (that is the more fragments field is zero) the total data length is computed. If this fragment completes the datagram (tested by checking the bits set in the fragment block table), then the datagram is sent to the next step in datagram processing; otherwise the timer is set to the maximum of the current timer value and the value of the time to live field from this fragment; and the reassembly routine gives up control.

If the timer runs out, the all reassembly resources for this buffer identifier are released. The initial setting of the timer is a lower bound on the reassembly waiting time. This is because the waiting time will be increased if the Time to Live in the arriving fragment is greater than the current timer value but will not be decreased if it is less. The maximum this timer value could reach is the maximum time to live (approximately 4.25 minutes). The current recommendation for the initial timer setting is 15 seconds. This may be changed as experience with

this protocol accumulates. Note that the choice of this parameter value is related to the buffer capacity available and the data rate of the transmission medium; that is, data rate times timer value equals buffer size (e.g., 10Kb/s X 15s = 150Kb).

Notation:

```
  FO    -  Fragment Offset
  IHL   -  Internet Header Length
  MF    -  More Fragments flag
  TTL   -  Time To Live
  NFB   -  Number of Fragment Blocks
  TL    -  Total Length
  TDL   -  Total Data Length
  BUFID -  Buffer Identifier
  RCVBT -  Fragment Received Bit Table
  TLB   -  Timer Lower Bound
```

Procedure:

```
  (1)  BUFID <- source|destination|protocol|identification;
  (2)  IF FO = 0 AND MF = 0
  (3)     THEN IF buffer with BUFID is allocated
  (4)             THEN flush all reassembly for this BUFID;
  (5)          Submit datagram to next step; DONE.
  (6)     ELSE IF no buffer with BUFID is allocated
  (7)             THEN allocate reassembly resources
                       with BUFID;
                       TIMER <- TLB; TDL <- 0;
  (8)          put data from fragment into data buffer with
               BUFID from octet FO*8 to
                                  octet (TL-(IHL*4))+FO*8;
  (9)          set RCVBT bits from FO
                                 to FO+((TL-(IHL*4)+7)/8);
  (10)         IF MF = 0 THEN TDL <- TL-(IHL*4)+(FO*8)
  (11)         IF FO = 0 THEN put header in header buffer
  (12)         IF TDL # 0
  (13)          AND all RCVBT bits from 0
                                     to (TDL+7)/8 are set
```

```
(14)            THEN TL <- TDL+(IHL*4)
(15)                 Submit datagram to next step;
(16)                 free all reassembly resources
                     for this BUFID; DONE.
(17)         TIMER <- MAX(TIMER,TTL);
(18)         give up until next fragment or timer expires;
(19) timer expires: flush all reassembly with this BUFID; DONE.
```

In the case that two or more fragments contain the same data

either identically or through a partial overlap, this procedure will use the more recently arrived copy in the data buffer and datagram delivered.

Identification

The choice of the Identifier for a datagram is based on the need to provide a way to uniquely identify the fragments of a particular datagram. The protocol module assembling fragments judges fragments to belong to the same datagram if they have the same source, destination, protocol, and Identifier. Thus, the sender must choose the Identifier to be unique for this source, destination pair and protocol for the time the datagram (or any fragment of it) could be alive in the internet.

It seems then that a sending protocol module needs to keep a table of Identifiers, one entry for each destination it has communicated with in the last maximum packet lifetime for the internet.

However, since the Identifier field allows 65,536 different values, some host may be able to simply use unique identifiers independent of destination.

It is appropriate for some higher level protocols to choose the identifier. For example, TCP protocol modules may retransmit an identical TCP segment, and the probability for correct reception would be enhanced if the retransmission carried the same identifier as the original transmission since fragments of either datagram could be used to construct a correct TCP segment.

Type of Service

The type of service (TOS) is for internet service quality selection. The type of service is specified along the abstract parameters precedence, delay, throughput, and reliability. These abstract parameters are to be mapped into the actual service parameters of the particular networks the datagram traverses.

Precedence. An independent measure of the importance of this datagram.

Delay. Prompt delivery is important for datagrams with this indication.

Throughput. High data rate is important for datagrams with this indication.

Reliability. A higher level of effort to ensure delivery is important for datagrams with this indication.

For example, the ARPANET has a priority bit, and a choice between "standard" messages (type 0) and "uncontrolled" messages (type 3), (the choice between single packet and multipacket messages can also be considered a service parameter). The uncontrolled messages tend to be less reliably delivered and suffer less delay. Suppose an internet datagram is to be sent through the ARPANET. Let the internet type of service be given as:

```
Precedence:    5
Delay:         0
Throughput:    1
Reliability:   1
```

In this example, the mapping of these parameters to those available for the ARPANET would be to set the ARPANET priority bit on since the Internet precedence is in the upper half of its range, to select standard messages since the throughput and reliability requirements are indicated and delay is not. More details are given on service mappings in "Service Mappings" [8].

Time to Live

The time to live is set by the sender to the maximum time the datagram is allowed to be in the internet system. If the datagram is in the internet system longer than the time to live, then the datagram must be destroyed.

This field must be decreased at each point that the internet header is processed to reflect the time spent processing the datagram. Even if no local information is available on the time actually spent, the field must be decremented by 1. The time is measured in units of seconds (i.e. the value 1 means one second). Thus, the maximum time to live is 255 seconds or 4.25 minutes. Since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Some higher level reliable connection protocols are based on assumptions that old duplicate datagrams will not arrive after a certain time elapses. The TTL is a way for such protocols to have an assurance that their assumption is met.

Options

The options are optional in each datagram, but required in implementations. That is, the presence or absence of an option is the choice of the sender, but each internet module must be able to parse every option. There can be several options present in the option field.

The options might not end on a 32-bit boundary. The internet header must be filled out with octets of zeros. The first of these would be interpreted as the end-of-options option, and the remainder as internet header padding.

Every internet module must be able to act on every option. The Security Option is required if classified, restricted, or compartmented traffic is to be passed.

Checksum

The internet header checksum is recomputed if the internet header is changed. For example, a reduction of the time to live, additions or changes to internet options, or due to fragmentation. This checksum at the internet level is intended to protect the internet header fields from transmission errors.

There are some applications where a few data bit errors are acceptable while retransmission delays are not. If the internet protocol enforced data correctness such applications could not be supported.

Errors

Internet protocol errors may be reported via the ICMP messages [3].

3.3. Interfaces

The functional description of user interfaces to the IP is, at best, fictional, since every operating system will have different

facilities. Consequently, we must warn readers that different IP implementations may have different user interfaces. However, all IPs must provide a certain minimum set of services to guarantee that all IP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all IP implementations.

Internet protocol interfaces on one side to the local network and on the other side to either a higher level protocol or an application program. In the following, the higher level protocol or application

program (or even a gateway program) will be called the "user" since it is using the internet module. Since internet protocol is a datagram protocol, there is minimal memory or state maintained between datagram transmissions, and each call on the internet protocol module by the user supplies all information necessary for the IP to perform the service requested.

An Example Upper Level Interface

The following two example calls satisfy the requirements for the user to internet protocol module communication ("=>" means returns):

SEND (src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt => result)

where:

```
src = source address
dst = destination address
prot = protocol
TOS = type of service
TTL = time to live
BufPTR = buffer pointer
len = length of buffer
Id  = Identifier
DF = Don't Fragment
opt = option data
result = response
  OK = datagram sent ok
  Error = error in arguments or local network error
```

Note that the precedence is included in the TOS and the security/compartment is passed as an option.

RECV (BufPTR, prot, => result, src, dst, TOS, len, opt)

where:

```
BufPTR = buffer pointer
prot = protocol
result = response
  OK = datagram received ok
  Error = error in arguments
len = length of buffer
src = source address
dst = destination address
TOS = type of service
opt = option data
```


When the user sends a datagram, it executes the SEND call supplying all the arguments. The internet protocol module, on receiving this call, checks the arguments and prepares and sends the message. If the arguments are good and the datagram is accepted by the local network, the call returns successfully. If either the arguments are bad, or the datagram is not accepted by the local network, the call returns unsuccessfully. On unsuccessful returns, a reasonable report must be made as to the cause of the problem, but the details of such reports are up to individual implementations.

When a datagram arrives at the internet protocol module from the local network, either there is a pending RECV call from the user addressed or there is not. In the first case, the pending call is satisfied by passing the information from the datagram to the user. In the second case, the user addressed is notified of a pending datagram. If the user addressed does not exist, an ICMP error message is returned to the sender, and the data is discarded.

The notification of a user may be via a pseudo interrupt or similar mechanism, as appropriate in the particular operating system environment of the implementation.

A user's RECV call may then either be immediately satisfied by a pending datagram, or the call may be pending until a datagram arrives.

The source address is included in the send call in case the sending host has several addresses (multiple physical connections or logical addresses). The internet module must check to see that the source address is one of the legal address for this host.

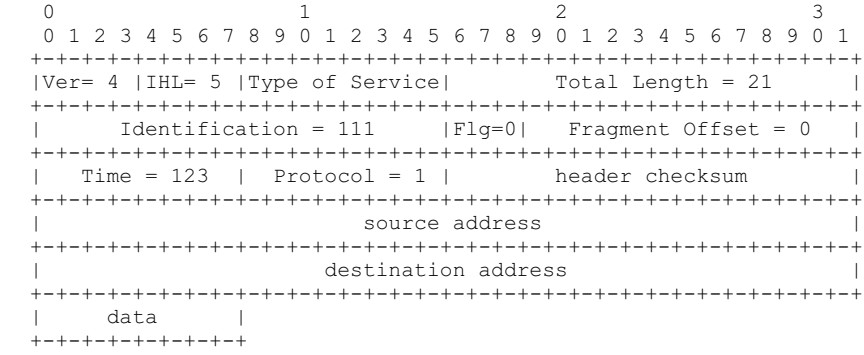
An implementation may also allow or require a call to the internet module to indicate interest in or reserve exclusive use of a class of datagrams (e.g., all those with a certain value in the protocol field).

This section functionally characterizes a USER/IP interface. The notation used is similar to most procedure of function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs), or any other form of interprocess communication.

APPENDIX A: Examples & Scenarios

Example 1:

This is an example of the minimal data carrying internet datagram:



Example Internet Datagram

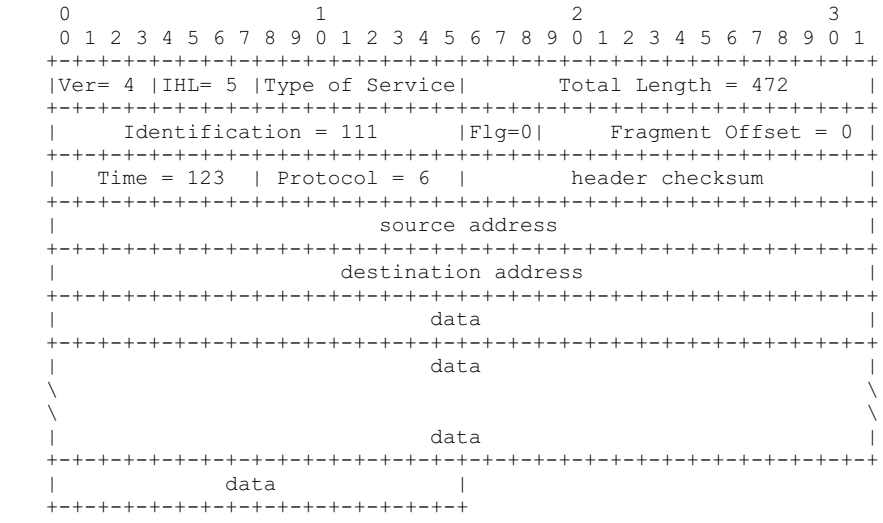
Figure 5.

Note that each tick mark represents one bit position.

This is a internet datagram in version 4 of internet protocol; the internet header consists of five 32 bit words, and the total length of the datagram is 21 octets. This datagram is a complete datagram (not a fragment).

Example 2:

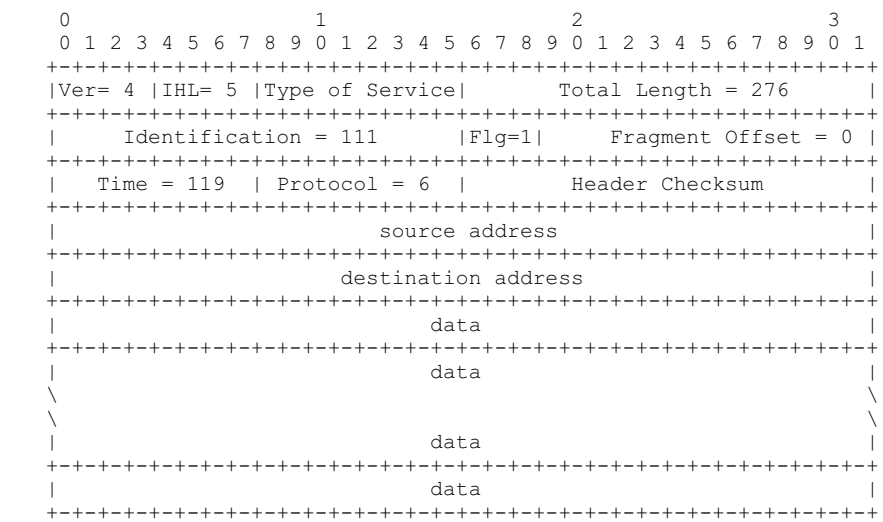
In this example, we show first a moderate size internet datagram (452 data octets), then two internet fragments that might result from the fragmentation of this datagram if the maximum sized transmission allowed were 280 octets.



Example Internet Datagram

Figure 6.

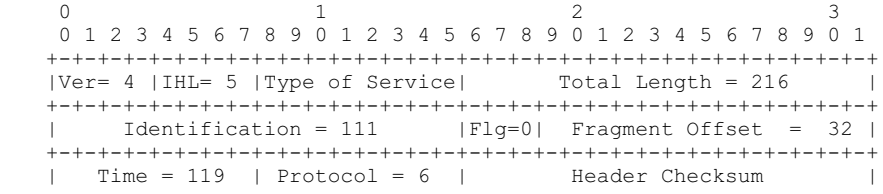
Now the first fragment that results from splitting the datagram after 256 data octets.

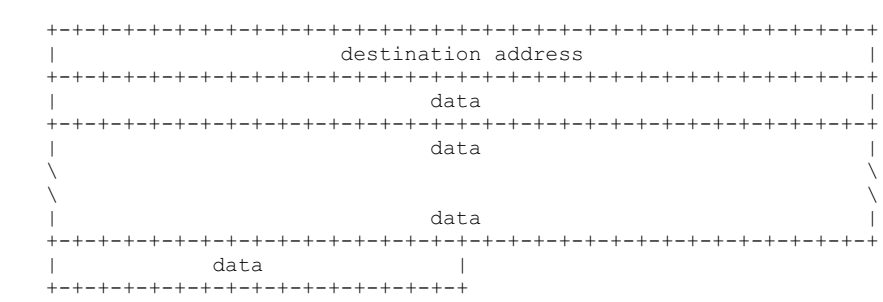


Example Internet Fragment

Figure 7.

And the second fragment.



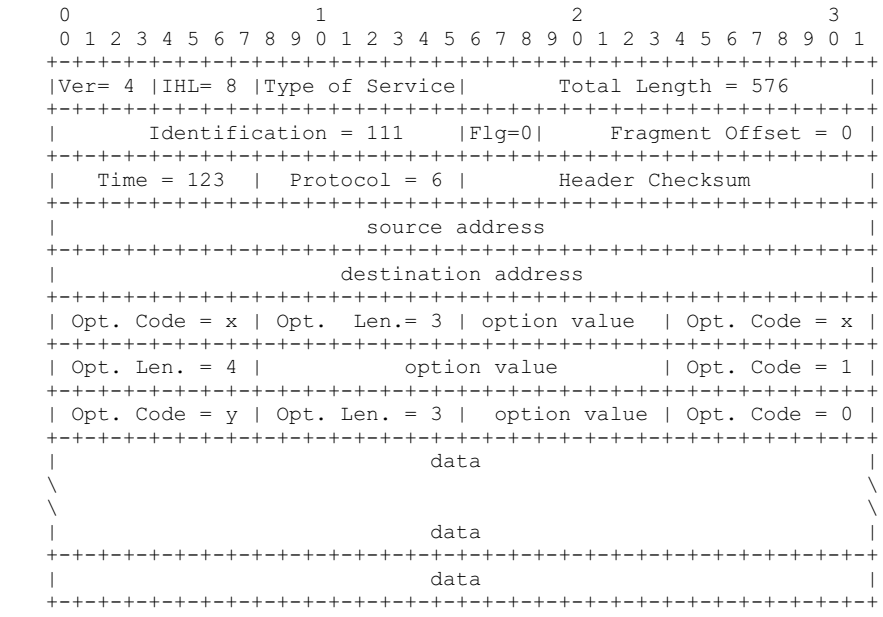


Example Internet Fragment

Figure 8.

Example 3:

Here, we show an example of a datagram containing options:

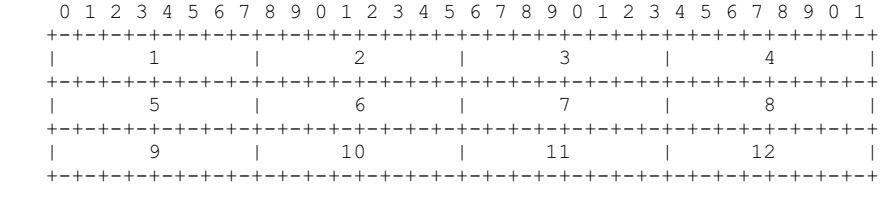


Example Internet Datagram

Figure 9.

APPENDIX B: Data Transmission Order

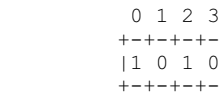
The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in the following diagram the octets are transmitted in the order they are numbered.



Transmission Order of Bytes

Figure 10.

Whenever an octet represents a numeric quantity the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit. For example, the following diagram represents the value 170 (decimal).



Significance of Bits

Figure 11.

Similarly, whenever a multi-octet field represents a numeric quantity the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet is transmitted first.

GLOSSARY

1822
: BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ARPANET leader
: The control information on an ARPANET message at the host-IMP interface.

ARPANET message
: The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet
: A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

Destination
: The destination address, an internet header field.

DF
: The Don't Fragment bit carried in the flags field.

Flags
: An internet header field carrying various control flags.

Fragment Offset
: This internet header field indicates where in the internet datagram a fragment belongs.

GGP
: Gateway to Gateway Protocol, the protocol used primarily between gateways to control routing and other gateway functions.

header
: Control information at the beginning of a message, segment, datagram, packet or block of data.

ICMP
: Internet Control Message Protocol, implemented in the internet module, the ICMP is used from gateways to hosts and between hosts to report errors and make routing suggestions.

Identification
An internet header field carrying the identifying value assigned by the sender to aid in assembling the fragments of a datagram.

IHL
The internet header field Internet Header Length is the length of the internet header measured in 32 bit words.

IMP
The Interface Message Processor, the packet switch of the ARPANET.

Internet Address
A four octet (32 bit) source or destination address consisting of a Network field and a Local Address field.

internet datagram
The unit of data exchanged between a pair of internet modules (includes the internet header).

internet fragment
A portion of the data of an internet datagram with an internet header.

Local Address
The address of a host within a network. The actual mapping of an internet local address on to the host addresses in a network is quite general, allowing for many to one mappings.

MF
The More-Fragments Flag carried in the internet header flags field.

module
An implementation, usually in software, of a protocol or other procedure.

more-fragments flag
A flag indicating whether or not this internet datagram contains the end of an internet datagram, carried in the internet header Flags field.

NFB
The Number of Fragment Blocks in a the data portion of an internet fragment. That is, the length of a portion of data measured in 8 octet units.

octet
An eight bit byte.

Options
The internet header Options field may contain several options, and each option may be several octets in length.

Padding
The internet header Padding field is used to ensure that the data begins on 32 bit word boundary. The padding is zero.

Protocol
In this document, the next higher level protocol identifier, an internet header field.

Rest
The local address portion of an Internet Address.

Source
The source address, an internet header field.

TCP

Transmission Control Protocol: A host-to-host protocol for reliable communication in internet environments.

TCP Segment
The unit of data exchanged between TCP modules (including the TCP header).

TFTP
Trivial File Transfer Protocol: A simple file transfer protocol built on UDP.

Time to Live
An internet header field which indicates the upper bound on how long this internet datagram may exist.

TOS
Type of Service

Total Length
The internet header field Total Length is the length of the datagram in octets including internet header and data.

TTL
Time to Live

Type of Service
An internet header field which indicates the type (or quality) of service for this internet datagram.

UDP
User Datagram Protocol: A user level protocol for transaction oriented applications.

User
The user of the internet protocol. This may be a higher level protocol module, an application program, or a gateway program.

Version
The Version field indicates the format of the internet header.

REFERENCES

[1] Cerf, V., "The Catenet Model for Internetworking," Information Processing Techniques Office, Defense Advanced Research Projects Agency, IEN 48, July 1978.

[2] Bolt Beranek and Newman, "Specification for the Interconnection of a Host and an IMP," BBN Technical Report 1822, Revised May 1978.

[3] Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification," RFC 792, USC/Information Sciences Institute, September 1981.

[4] Shoch, J., "Inter-Network Naming, Addressing, and Routing," COMPCON, IEEE Computer Society, Fall 1978.

[5] Postel, J., "Address Mappings," RFC 796, USC/Information Sciences Institute, September 1981.

[6] Shoch, J., "Packet Fragmentation in Inter-Network Protocols," Computer Networks, v. 3, n. 1, February 1979.

[7] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.

[8] Postel, J., "Service Mappings," RFC 795, USC/Information Sciences Institute, September 1981.

[9] Postel, J., "Assigned Numbers," RFC 790, USC/Information Sciences Institute, September 1981.

Comment on RFC 791

Comments about this RFC:

- RFC 791: RFC791 was analyzed by me including an assortment of conjectures from laplace... by Mark(god of waq) (3/1/2004)

Previous: RFC 0790 - Assigned numbers

Next: RFC 0792 - Internet Control Message Protocol

[RFC Index | RFC Search | Usenet FAQs | Web FAQs | Documents | Cities]

RFC 959 (RFC959)

Internet RFC/STD/FYI/BCP Archives

RFC 959 - File Transfer Protocol

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

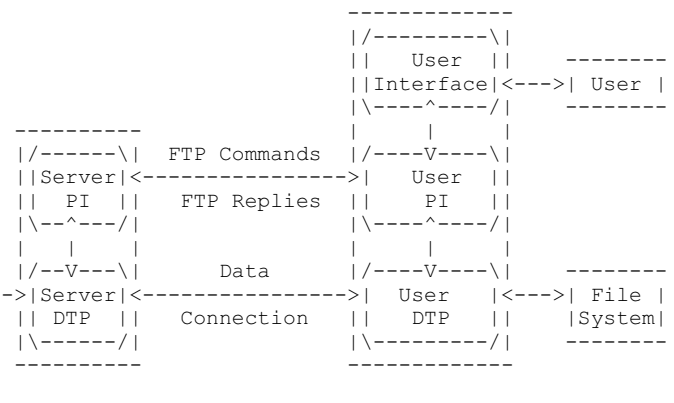
A set of functions including a protocol interpreter, a data transfer process and a user interface which together perform the function of file transfer in cooperation with one or more server-FTP processes. The user interface allows a local language to be used in the command-reply dialogue with the user.

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.

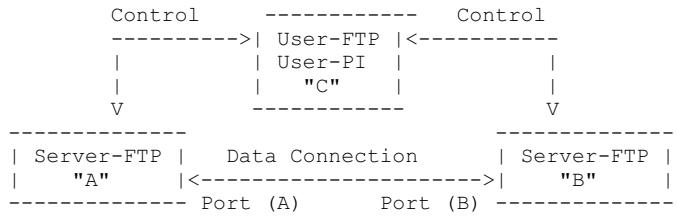


Figure 2

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.

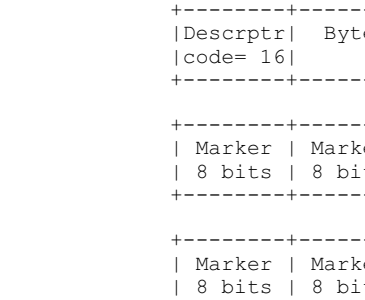
Code	Meaning
128	End of data block is EOR

```
     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.

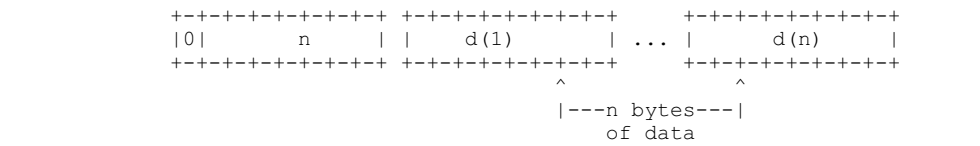
For example, to transmit a six-character marker, the following would be sent:



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.

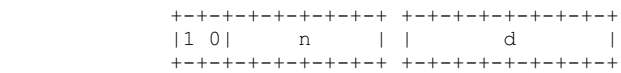
Byte string:



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

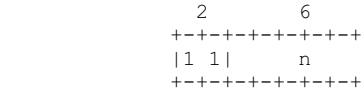
To compress a string of n replications of the data byte d, the following 2 bytes are sent:

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.

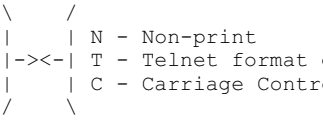
PASSIVE (PASV)

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
```

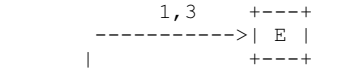

6. STATE DIAGRAMS

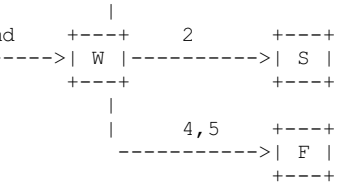
Here we present state diagrams for a very simple minded FTP implementation. Only the first digit of the reply codes is used. There is one state diagram for each group of FTP commands or command sequences.

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:

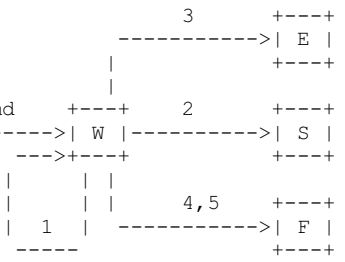




This diagram models the commands:

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:

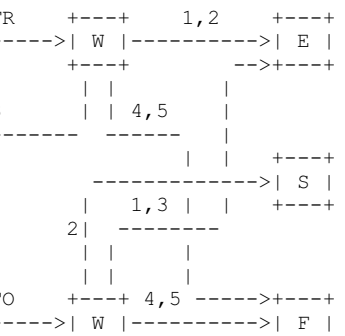


This diagram models the commands:

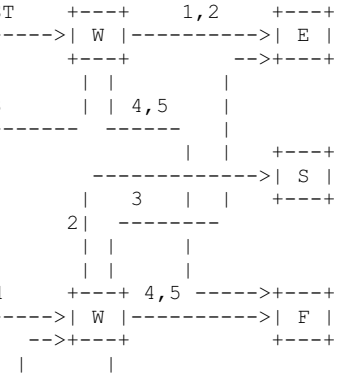
APPE, LIST, NLST, REIN, RETR, STOR, and STOU.

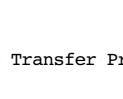
Note that this second model could also be used to represent the first group of commands, the only difference being that in the first group the 100 series replies are unexpected and therefore treated as error, while the second group expects (some may require) 100 series replies. Remember that at most,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

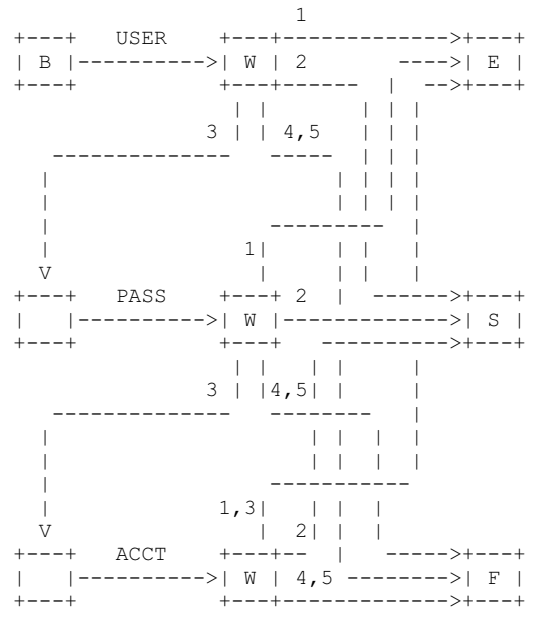




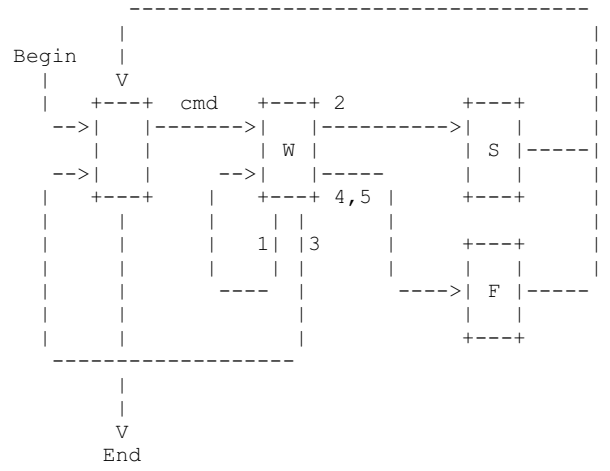
Where "cmd" is APPE, STOR, or RETR.

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.

Comment on RFC 959

Comments about this RFC:

- RFC 959: it is very nice please send me the codings in java to create an ftp application... by chandru (1/12/2004)
- RFC 959: I am very much intrested in this area of reasearch work. I am also implementing... by SatyanandamN (5/14/2004)
- RFC 959: RFC 959 does not seem to be working with directory names (<pathname>) that... by vpogrebi (7/15/2004)
- RFC 959: Article is good. If Possible plz send me the Java Code for applying FTP to my... by ravish (2/29/2004)
- RFC 959: "Directory Oriented FTP Commands" is RFC 775, not 776 as stated in this RFC. by Maury Markowitz (4/28/2004)
- RFC 959: I am very much intrested in C and C++ on linux platform. If u have any documenta... by puram (6/2/2004)
- RFC 959: Its very nice to see. Please send me ftp client-server related documentatioin... by Himanshu Agrawal (12/7/2003)
- RFC 959: Someone can send me some java code for realize ftp server??? the adrress is... by merlo (6/10/2004)
- RFC 959: Well this artile is fabulous i need to have a look on ftp implementation in C or... by Namna (6/12/2004)

Previous: RFC 0958 - Network Time Protocol (NTP) Next: RFC 0960 - Assigned numbers

[RFC Index | RFC Search | Usenet FAQs | Web FAQs | Documents | Cities]