Advanced Streaming Format (ASF) Specification

Status of This Memo

This document is an Internet-Draft. Internet-Drafts are working
documents of the Internet Engineering Task Force (IETF), its areas, and
its working groups. Note that other groups may also distribute working
documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months
and may be updated, replaced, or obsoleted by other documents at any
time. It is inappropriate to use Internet-Drafts as reference material
or to cite them other than as ``work in progress.''

To learn the current status of any Internet-Draft, please check the
``1id-abstracts.txt'' listing contained in the Internet-Drafts Shadow
Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe),
munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or
ftp.isi.edu (US West Coast).

Distribution of this document is unlimited.

Abstract

The Advanced Streaming Format (ASF) is an extensible file format
designed to store synchronized multimedia data. It supports data
delivery over a wide variety of networks and protocols while still
proving suitable for local playback. ASF supports advanced multimedia
capabilities including extensible media types, component download,
scaleable media types, author-specified stream prioritization, multiple
language support, and extensive bibliographic capabilities, including
document and content management.

Table of Contents

1 **Introduction**

1.1 **What is ASF?**

The Advanced Streaming Format (ASF) is an extensible file format
designed to store synchronized multimedia data. It supports data
delivery over a wide variety of networks and protocols while still
proving suitable for local playback. The explicit goal of ASF is to
provide a basis for industry-wide multimedia interoperability, with ASF
being adopted by all major streaming solution providers and multimedia
authoring tool vendors.

Each ASF file is composed of one or more media streams. The file
header specifies the properties of the entire file, along with stream-
specific properties. Multimedia data, stored after the file header,
references a particular media stream number to indicate its type and
purpose. The delivery and presentation of all media stream data is
synchronized to a common timeline.

The ASF file definition includes the specification of some commonly
used media types (see Section 8). The explicit intention is that if an
implementation supports media types from within this set of standard
media types (in other words, audio, video, image, timecode, text, MIDI,
command, or media object), then that media type must be supported in
the manner described in Section 8 if the resulting content is to be
considered to be "content compliant" with the ASF specification.
Implementations are free to support other media types (in addition to
the currently defined standard media types) in any way they see fit.

Finally, ASF is said to support the transmission of "live content" over
a network. This refers to multimedia content that may or may not ever
become recorded upon a persistent media source (for example, a disk,
CD-ROM, DVD, etc). This use explicitly and solely means that
information describing the multimedia content must have been received
before the multimedia data itself is received (in order to interpret
the multimedia data), and that this information must convey the
semantics of the ASF Header Object. Similarly, the received data must
correspond to the format of the ASF data units. No additional information
should be conveyed by this term. Specifically, this use explicitly does
not refer to (or contain) any information about network control
protocols or network transmission protocols. It refers solely to the
order of information arrival (header semantics before data) and the
data format.

1.2 **Design Goals**

ASF was designed with the following goals:

* To support efficient playback from media servers, HTTP servers, and
  local storage devices.
* To support scalable media types such as audio and video.
* To permit a single multimedia composition to be presented over a
  wide range of bandwidths.
* To allow authoring control over media stream relationships,
  especially in constrained-bandwidth scenarios.
* To be independent of any particular multimedia composition system,
  computer operating system, or data communications protocol.

1.3 Scope

ASF is a multimedia presentation file format. It supports live and on-
demand multimedia content. It can be used as a vehicle to record or
play back H.32X (for example, H.323 and H.324) or MBONE conferences.
ASF files may be edited. ASF data is specifically designed for
streaming and/or local playback.

ASF is not:
* ASF is not a wire format. Rather, ASF is data communications
  "agnostic." Theoretically, ASF data units may be carried by any
  conceivable underlying data communications transport. ASF is
  similarly agnostic about how the data is packetized by network
  protocols (for example, whether the multimedia data is sent in an
  interleaved or non-interleaved fashion).
* ASF is not a network control protocol. However, ASF files contain
  information that should prove useful to control protocols.
* ASF is not a replacement for MPEG. Rather, MPEG content can be
  contained within ASF files and optionally synchronized with other
  media.

2 ASF Features

2.1 Extensible Media Types

ASF files permit authors to easily define new media types. The ASF
format provides sufficient flexibility to allow the definition of new
media stream types that conform to the file format definition. Each
stored media stream is logically independent from all others unless a
relationship to another media stream has been explicitly established in
the file header.

2.2 Component Download

Stream-specific information about playback components (for example,
decompressors and renderers) can be stored in the file header. This
information enables each client implementation to retrieve the

appropriate version of the required playback component if it is not
already present on the client machine.

2.3 Scalable Media Types

ASF is designed to express the dependency relationships between logical
"bands" of scalable media types. It stores each band as a distinct
media stream. Dependency information among these media streams is
stored in the file header, providing sufficient information for clients
to interpret scalability options (such as spatial, temporal, or quality
scaling for video) in a compression-independent manner.

2.4 Author-specified Stream Prioritization

Modern multimedia delivery systems can dynamically adjust to changing
constraints (for example, available bandwidth). Authors of multimedia
content must be able to express their preferences in terms of relative
stream priorities as well as a minimum set of streams to deliver.
Stream prioritization is complicated by the presence of scalable media
types, since it is not always possible to determine the order of stream
application at authoring time. ASF allows content authors to
effectively communicate their preferences, even when scalable media
streams are present.

2.5 Multiple Languages

ASF is designed to support multiple languages. Media streams can
optionally indicate the language of the contained media. This feature
is typically used for audio or text streams. A multilingual ASF file
indicates that a set of media streams contains different language
versions of the same content, allowing an implementation to choose the
most appropriate version for a given client.

2.6 Bibliographic Information

ASF provides the capability to maintain extensive bibliographic
information in a manner that is highly flexible and very extensible.
All bibliographic information is stored in the file header in Unicode
and is designed for multiple language support, if needed. Bibliographic
fields can either be predefined (for example, author and title) or
author-defined (for example, search terms). Bibliographic entries can
apply to either the whole file or a single media stream.

3 File Format Organization.

3.1 ASF Object definition

The base unit of organization for ASF files is called the ASF Object.
It consists of a 128-bit globally unique identifier (GUID) for the

object, a 64-bit integer object size, and variable length object data.
The value of the object size field is the sum of 24 bytes plus the size
of the object data in bytes.
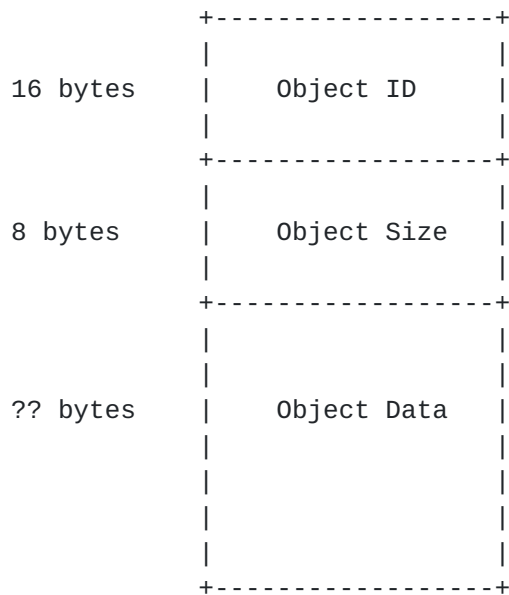
```
                     +------------------+
                     |                  |
        16 bytes     |     Object ID    |
                     |                  |
                     +------------------+
                     |                  |
         8 bytes     |    Object Size   |
                     |                  |
                     +------------------+
                     |                  |
                     |                  |
        ?? bytes     |    Object Data   |
                     |                  |
                     |                  |
                     |                  |
                     |                  |
                     +------------------+
```
            Figure 1: ASF Object

This unit of file organization is similar to the Resource Interchange
File Format (RIFF) chunk, which is the basis for AVI and WAV files.
The ASF object enhances the design of the RIFF chunk in two ways.
First, there is no need for a central authority to manage the object
identifier system, since any computer with a network card can generate
valid, unique GUIDs (see Appendix C). Second, the object size has been
chosen to be large enough to handle the very large files needed for
high-bandwidth multimedia content.

All ASF objects and structures (including data unit headers) are stored
in little-endian byte order (the inverse of network byte order).
However, ASF files can contain media stream data in either byte order
within the data unit.

3.2 High-level File Structure

ASF files are logically composed of three top-level objects: the Header
Object, the Data Object, and the Index Object. The Header Object is
mandatory and must be placed at the very beginning of every ASF file.
The Data Object is also mandatory, and should normally follow the Header
Object. The Index Object is optional, but it is strongly recommended that
it be used.

Implementations will support files containing out-of-order objects, but
in certain cases the resulting ASF files will not be usable from
certain sources such as HTTP servers. Also, additional top-level
objects may be defined by implementations and inserted into ASF files.
It is recommended that they follow the Index Object (in object
placement order).

A requirement of ASF is that the Header Object must have been received
for the contents of the Data Object to be interpreted. ASF does not
address how this information arrives at the client. Rather, "arrival
mechanisms" are deemed to be a "local implementation issue," which is
explicitly out of the scope of the file specification. It is similarly
a local implementation issue whether or not the Header Object is
transferred "in band" or "out of band" (vis-a-vis the Data Object's
data units) or whether the Header Object is sent once or is repeatedly
sent. Implementations may choose to meet this order requirement (in
other words, the Header Object must arrive before ASF data units can be
interpreted) in many possible ways including: (A) include the Header
Object information as part of the "session announcement"; (B) send the
Header Object in a different "channel" (e.g., link) than the data
object; (C) send the Header Object immediately before the ASF data
units; and so on.

```
              +---------------------------------+
              |          Header Object          |
              |   +---------------------------+  |
              |   | File Properties Object    |  |
              |   +---------------------------+  |
              |   | Stream Properties Object 1 |  |
              |   +---------------------------+  |
              |   | Stream Properties Object N |  |
              |   +---------------------------+  |
              |   |    Other Header Objects   |  |
              |   +---------------------------+  |
              +---------------------------------+
              |           Data Object           |
              |       +-------------------+     |
              |       |     Data Unit     |     |
              |       +-------------------+     |
              |       |     Data Unit     |     |
              |       +-------------------+     |
              |       |     Data Unit     |     |
              |       +-------------------+     |
              +---------------------------------+
              |           Index Object          |
              +---------------------------------+
              |          Other Objects          |
              +---------------------------------+
```
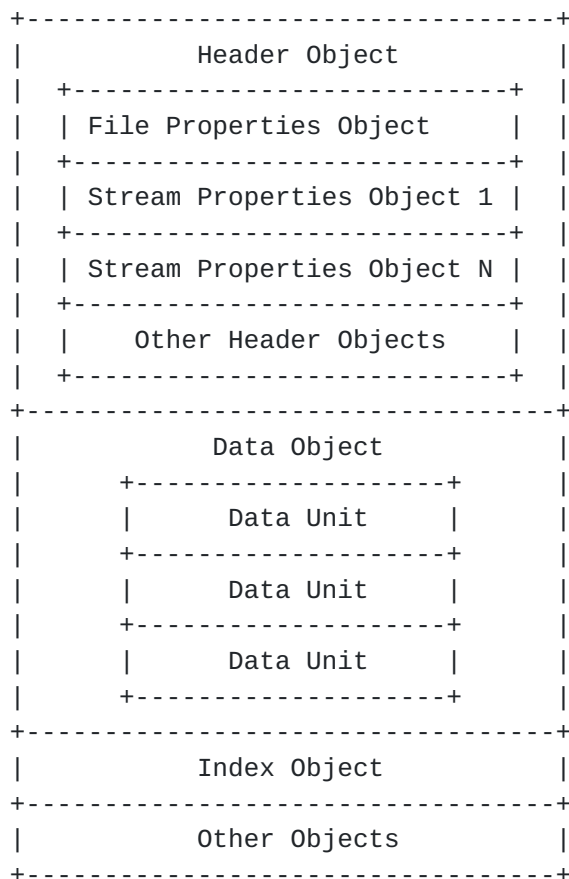
Figure 2. High-level ASF File Structure

3.3 ASF Header Object

Of the three top-level ASF objects, the Header Object is the only one
that contains other ASF objects. The header object may include many
objects including the following:
* File Properties Object - global file attributes
* Stream Properties Object - defines a media stream and its
  characteristics
* Content Description Object - contains all bibliographic information
* Component Download Object - provides playback component information
* Stream Groups Object - logically groups media streams together
* Scalable Object - defines scalability relationships among media
  streams containing bands
* Prioritization Object - defines relative stream prioritization
* Mutual Exclusion Object - defines exclusion relationships such as
  language selection
* Inter-Media Dependency Object - defines dependency relationships
  among mixed media streams
* Rating Object - defines the Rating of the file in terms of W3C PICS
* Index Parameters Object - supplies the information necessary to
  regenerate the index of an ASF file

The role of the Header Object is to provide a well-known byte sequence
at the beginning of ASF files (its GUID) and to contain all other
header information. This information provides global information about
the file as a whole as well as specific information about the
multimedia data stored within the Data Object.

3.4 ASF Data Object

The Data Object contains all the multimedia data of an ASF file. This
data is stored in the form of ASF data units. Each ASF Data Unit is of
variable length, and contains data for only one media stream. Data
units are sorted within the Data Object based on the time at which they
should be delivered (send time). This sorting results in an
interleaved data format.

3.5 ASF Index Object

The Index Object contains a time-based index into the multimedia data
of an ASF file. The time interval that each index entry represents is
set at authoring time and stored in the Index Object. Since it is not

required to index into every media stream in a file, a list of the
media streams that are indexed follows the time interval value.

Each index entry consists of one data unit offset per media stream being
indexed. This information allows
stream-specific index operations to occur.

3.6 Minimal Implementation

A minimal ASF implementation consists of a Header Object containing
only a File Properties Object, one Stream Properties object, and one
Language List Object, as well as a Data Object containing only a single
ASF data unit.

4 Additional Considerations

4.1 Time Units

All time fields in ASF objects and ASF data units use the same
timeline, which begins at time zero. Send Times (see Section 4.2) are
expressed in granularities of milliseconds. Presentation Times (see
Section 4.2) are expressed in Rational Time units. Other timecode
systems (such as SMPTE) are supported through the use of a timecode
media stream that binds alternate timecode values to each data unit
(see Section 8.4). This stream binding is achieved using the Inter-
Media Dependency Object. This allows authoring and editing tools to
keep alternate timestamps while permitting client/server
implementations to ignore them. In all cases, all time references are
to the same timeline.

4.2 Send Time vs. Presentation Time

ASF Data Units all contain a millisecond timestamp, which is called the
data unit's send time. This is the time on the ASF timeline at which
this data unit should be delivered to the client. Sometimes, the media
stream can explicitly store the fixed delta between send time and
presentation time in the Stream Properties Object. If so, every data
unit for that stream is presented at exactly the same amount of time
after being sent. If this delta is zero, then the send time is
equivalent to the presentation time. Otherwise, the data unit stores
the presentation time in the data unit itself as either a delta value
from the send time or as an explicit presentation timestamp. Using data
unit-specific presentation times provides increased flexibility to
authoring tools to reduce a stream's maximum bandwidth requirement by
sending data before it is needed.

Unlike Send Time, Presentation Time is specified in Rational Time
units, thereby permitting finer time granularities than is possible for

millisecond units. The numerator and denominator values by which the
specific Rational Time units are computed for each media stream are
established in that media stream's Stream Properties Object.

4.3 Scalable Media Types

Information about each scalable media source (for example, audio or
video) is stored in a Scalable Object in the header. If multiple types
of scalable media are present in one ASF file, the header will contain
multiple Scalable Objects.

Each Scalable Object contains the dependency information for all media
streams that comprise bands of the same media source. Also included
within the Scalable Object is an author-specified default sequence in
which the media stream bands should be applied. This information is
useful if a client is unable or unwilling to resolve the user's
scalability preferences. The sequence also specifies the enhancement
type of each media stream band. For scalable video, there are three
common enhancement types: spatial (increasing frame size), temporal
(increasing frame rate), and quality (increasing image quality without
resizing). Similarly, scalable audio has number of channels (for
example, stereo), frequency response, and quality. Additional user-
defined enhancement types may also be defined.

4.4 Multimedia Composition

One of ASF's design goals is to be independent of any particular
multimedia composition system. No information is provided in the ASF
format concerning three-dimensional positions of streams or relative
positioning information between streams. Using the Stream Group Object,
ASF provides a general mechanism to group logically related media
streams. Implementations will then determine how to render these
streams (for example, the relative positioning of the grouped streams,
stream mixing, Z-ordering and all other compositional issues, etc) by a
mechanism that is outside scope of this file specification. This
determination may be based on "out-of-band" techniques such as end user
input, the client environment itself, or information contained within
the media streams themselves (for example, MPEG-4, streaming Dynamic
HTML content, and so on.).

It is anticipated that several different composition approaches can
coexist and leverage the same piece of ASF content. An example is a TV
scenario in which two video streams are grouped separately. One
contains a large image of the anchorperson against a backdrop, and the
other contains smaller footage of a news story. While the size of each
rendering site could be calculated based on the natural size of each
video stream in the group, the fact that the news story should be
overlaid on the top right corner of the anchorperson video can not be
determined without external composition information.

5 **ASF Header Object**

This section defines the various objects that comprise the ASF Header
Object.

5.1 **Header Object**

Mandatory: Yes
Quantity: 1 only

```
+------------------+----------------+--------------+
|    Field Name    |   Field Type   |  Size (bits) |
+------------------+----------------+--------------+
|    Object ID     |     GUID       |     128      |
|   Object Size    |     UINT       |      64      |
+------------------+----------------+--------------+
```

Notes:
The Header Object is a container that can hold any combination of the
following standard objects. Only the File Properties Object and the
Stream Properties Object are required to be present. In addition, (non-
standard) header objects that conform to the ASF Object Structure (see
Section 3.1) may also be optionally defined and used as extension
mechanisms for local implementations. Unlike the standard header
objects defined below, there is no guarantee that the non-standard
objects will be interpretable across vendor implementations.
Implementations should ignore any non-standard object that they do not
understand.

5.2 **File Properties Object**

Mandatory: Yes
Quantity: 1 only

This object defines the global characteristics of the combined media
streams found within the Data Object.

Object Structure:
```
+-------------------------+-----------------+---------------+
|       Field Name        |   Field Type    |  Size (bits)  |
+-------------------------+-----------------+---------------+
| Object ID               |      GUID       |      128      |
| Object Size             |      UINT       |       64      |
| File ID                 |      GUID       |      128      |
| Creation Date           |    FILETIME     |       64      |
| Content Expiration Date |    FILETIME     |       64      |
| Last Send Time          |      UINT       |       64      |
```

```
| Play Duration           |      UINT       |      64       |
| Flags                   |      UINT       |      32       |
-----+                    +-----------------+---------------+
     | Live Flag          |                 |    1 (LSB)    |
     |Huge Data Units Flag|                 |      1        |
     | Reserved           |                 |      30       |
+----+--------------------+-----------------+---------------+
| Minimum Bitrate         |      UINT       |      32       |
| Maximum Bitrate         |      UINT       |      32       |
| Average Data Unit Size  |      UINT       |      32       |
| Maximum Data Unit Size  |      UINT       |      32       |
| Total Data Units        |      UINT       |      32       |
| Stream Count            |      UINT       |      16       |
+-------------------------+-----------------+---------------+
```

Notes:
The Object ID field is the GUID for the File Properties Object (see
Appendix A). The Object Size field is the size (in bytes) of the File
Properties Object.

The value of the File ID field should be regenerated every time the
file is edited. It provides a unique identification for this ASF file.

The Creation Date contains the date and time of the initial creation of
the file.

Content Expiration Date indicates the date after which the author
doesn't want the file to be used. This time can be "never" (value of
zero).

Both the Last Send Time and the Play Duration fields have millisecond
granularities. Both of these fields are invalid if the live Flag bit is
set. Last Send Time is the send time of the last data unit within the
file. Play Duration is the maximum End Time (of any of the SPOs) minus the
minimum Start Time (of any of the SPOs).

The following are the meanings of the Flags:
* The Live Flag, if set, indicates that a file is in the process of
  being written (for example, for recording applications), and
  therefore various values stored in the header objects are invalid.
  It is highly recommended that post-processing be performed to remove
  this condition at the earliest opportunity.
* The Huge Data Units Flag determines whether the Data Unit Length
  field in the ASF Data Unit (Section 6.1) is 16 or 32 bits long (in
  other words, 0 signifies 16 bits, and 1 signifies 32 bits). The 32-
  bit Data Unit Length field should be used exclusively for local

recording/editing at extremely high data rates. Any other use is
strongly discouraged, since most networks will not be able to
support such huge data units. Therefore, it is strongly recommended
that the 16-bit Data Unit Length field alternative be used in the
general case.

Minimum Bit Rate is in bits per second and indicates the total of the
average bandwidth of all the mandatory streams.

Maximum Bit Rate is in bits per second and indicates the total of the
maximum bandwidth of all of the non-excluded streams.

The Average Data Unit Size is in bytes. This field is invalid if the
Live Flag is set.

The Maximum Data Unit Size is in bytes. This indicates the longest ASF
Data Unit within the Data Object. This field is invalid if the Live
Flag is set.

The Total Data Units field contains the number of ASF Data Unit entries
that exist within the Data Object. This field is invalid if the Live
Flag is set.

Stream Count field indicates the number of Stream Properties Objects
(SPOs) that exist in this file. Each media stream is required to have
its own SPO.

Invalid fields should have a value of zero for writing and should be
ignored when reading.

5.3 Stream Properties Object

Mandatory: Yes
Quantity: 1 per media stream

This object defines the specific properties and characteristics of a
media stream. It defines how a multimedia stream within the Data Object
is to be interpreted as well as the specific format (of elements) of
the ASF Data Unit itself (see Section 6.1) for that media stream. One
instance of this object is required for each media stream in the file,
including each of the separate streams formed by a scalable media type.

Unlike most other ASF objects, the Stream Properties Object (SPO) is a
"container object": it can optionally include additional ASF Objects
(see Section 3.1) within itself in a manner similar to the Header
Object. The size of these objects is included within the Object Size
field and contained objects, if any, are appended after the Type-

Specific Data field within the object structure below. This provision
dramatically enhances the scalability and expandability capabilities of
ASF, since it permits the rapid introduction of innovations and support
for technology evolution. Currently, only one ASF Object targeted to be
optionally contained within the SPO has been defined within this
specification: the Data Unit Extension Object (See Section 5.3.1).
Other ASF objects (for example, alternative approaches to scalable
media, a QoS (RSVP) information object, extra RTP information, or MPEG-
4 enhancements) may subsequently be defined and included within the SPO
as needed. In this way the SPO can be enhanced over time to embrace new
technologies and innovations.

Object Structure:

Field Name	Field Type	Size (bits)
Object ID	GUID	128
Object Size	UINT	64
Stream Type	GUID	128
Start Time	UINT	64
End Time	UINT	64
Average Bitrate	UINT	32
Maximum Bitrate	UINT	32
Average Data Unit Size	UINT	32
Maximum Data Unit Size	UINT	32
Preroll	UINT	32
Flags	UINT	32
Reliable Flag		1 (LSB)
Recordable Flag		1
Seekable Flag		1
Presentation Time Flags		2
Reserved		27
Presentation Time Delta	UINT	0 or 32
Presentation Time Numerator	UINT	0 or 32
Presentation Time Denominator	UINT	0 or 32
Stream Number	UINT	16
Stream Language ID Index	UINT	16
Stream Name Count	UINT	16
Stream Names	See below	?
MIME Type Length	UINT	8
MIME Type	ASCII (UINT8)	?
Type-Specific Data Length	UINT	16
Type-Specific Data	UINT8	?

Stream Name:

+-----------------------------+-----------------+---------------+
| Field Name | Field Type | Size (bits) |
+-----------------------------+-----------------+---------------+
Language ID Index	UINT	16
Stream Name Length	UINT	16
Stream Name	Unicode (UINT16)	?
+-----------------------------+-----------------+---------------+

Notes:
The Object ID field is the GUID for the Stream Properties Object (see
Appendix A). The Object Size field is the size (in bytes) of this Stream
Properties Object instance (including the sizes of all contained objects).

Start Time and End Time are presentation times in millisecond granularities.
Both fields are invalid if the Live Flag of the File Properties Object has
been set. The Start Time is the presentation time of the first object. The
End Time is the presentation time of the last object plus the duration of
play. The time reference in both cases is relative to the the ASF file's
timeline. These fields exist, therefore, to indicate where this media stream
is located within the context of the timeline of the file as a whole.

Invalid fields should have a value of 0 (zero) for writing and should be
ignored when reading.

The Average Bit Rate and the Maximum Bit Rates are in bits per second. Both
fields solely refer to this media stream's Bit Rates. The Maximum Bit Rate
is computed by identifying the maximum rate in any one-second period. The
Maximum Bit Rate means that the Bit Rate for this stream must not ever exceed
this value. This may be thought of as running a one second "sliding window"
over this media stream's contents and noting the specific one second interval
in which the greatest number of bits-per-second occurred. This value must be
non-zero. The Average Bit Rate is the approximation one would obtain by
dividing the total bits sent within this media stream by the time (in
seconds) during which those bits are being sent (i.e., one plus the send
time of the last Data Unit of that stream minus the send time of first data
unit of that stream).

The Average Data Unit Size and the Maximum Data Unit Size are in bytes and
refer to the ASF Data Units for this media's data types within the Data
Object. The Average Data Unit Size is computed by dividing the total size
of all of the ASF Data Units of that stream by the number of ASF Data Units
of that stream. The Maximum Data Unit Size is the size in bytes of the largest

ASF-DU for this media stream. A value of zero means "unknown". These values
are aids to the server for making network fragmentation and packetization
decisions.

Preroll is the minimum delay factor in milliseconds that a client should
use between starting a particular stream and starting the clock for the
client's timeline. It is used to compute the buffering requirements at
the client in order to mitigate against network jitter. Specifically,
when a data unit is received whose send time value is greater than the
preroll value for that stream, the client's timeline clock is started.
Rendering is subsequently determined by the Data Unit's presentation time
for that (i.e., the client's) timeline. The default preroll value is zero.

The following is the significance of the various flags in the Flags
field:
* Setting the Reliable Flag signifies that this media stream, if sent
  over a network, must be carried over a reliable data communications
  transport mechanism.
* Setting the Recordable Flag signifies that the content author has
  given permission for this media stream to be recorded. "Recorded,"
  in this context, means that the client system can preserve the
  content for later end-user use by writing that content to a place
  (for example, a disk, CD-ROM, and DVD) where the end user can later
  access it. The Recordable Flag should be set unless the author
  explicitly does not want the material to be recorded.
* Setting the Seekable Flag means that this media stream may be
  presented starting at a non-zero time offset. This implies that
  this stream is a potential candidate to be included within an index
  since the media stream may be correctly understood - and potentially
  played -- from additional locations other than only the stream's
  beginning.
* The Presentation Time Flags are 2 bits long, signifying the
  following:

```
  +-------+----------+---------------------------------------------+
  | Value:| Meaning: |  Explanation:                               |
  +-------+----------+---------------------------------------------+
  |   00  | Not Used | The Presentation Time field is not used     |
  |       |          | within the ASF Data Unit (see Section 6.1)  |
  |       |          | for this media stream. The Presentation     |
  |       |          | Time Delta, Presentation Time Numerator,    |
  |       |          | and the Presentation Time Denominator       |
  |       |          | fields are also not used within this object.|
  +-------+----------+---------------------------------------------+
  |   01  |  Fixed   | The Presentation Time field is not used     |
  |       |  Delta   | within the ASF Data Unit (see Section       |
  |       |          | 6.1) for this media stream. However,        |
  |       |          | the presentation time is known to be        |
```

```
|       |          | a fixed delta (in Rational Units) off of   |
|       |          | the send time. This delta is established   |
|       |          | by the Presentation Time Delta field       |
|       |          | within this object (in other words, this   |
|       |          | is the only case in which the Presentation |
|       |          | time Delta field is used within this object)|
+-------+----------+--------------------------------------------+
|   10  | Delta in | A 16-bit Presentation Time field (in       |
|       |Data Units| Rational Units) is used within the ASF     |
|       |          | Data Unit (see Section 6.1) for this       |
|       |          | media stream. That field identifies         |
|       |          | the presentation time as a delta off of     |
|       |          | the send time. The Presentation Time       |
|       |          | Delta field is not used within this object. |
+-------+----------+--------------------------------------------+
|   11  | Full Data| A 32-bit Presentation Time field (in       |
|       | Unit Pre-| Rational Units) is used within the ASF Data |
|       | sentation| Unit (see Section 6.1)for this media stream.|
|       |    Time  | That field identifies the actual            |
|       |          | presentation time for that data unit. The   |
|       |          | Presentation Time Delta field is not used   |
|       |          | within this object.                         |
+-------+----------+--------------------------------------------+
```

The Presentation Time Delta, Presentation Time Numerator, and
Presentation Time Denominator fields do not exist if the Presentation
Time Flags have a zero value. The Presentation Time Delta field also
does not exist if the Presentation Time Flags have 10 or 11 values (in
other words, it only exists if the flags have an 01 value; see above).
Otherwise these fields are 32 bits long.

Presentation Time Delta is in Rational Time Units. It indicates that a
fixed time delta (in Rational Units) between the presentation time and
the send time should be applied to the entirety of this stream's data
units (see the ASF Data Unit definition in Section 6.1). The
Presentation Time flags determine whether or not this field is used.

Rational Time Units signify a media-stream specific time unit within
the ASF file's intrinsic timeline. Rational Time Units are for
Presentation Times only. They are determined by dividing the
Presentation Time Numerator by the Presentation Time Denominator. The
default Presentation Time Numerator value is 1 and the default
Presentation Time Denominator value is 1000. Therefore, the default
Rational Time Units are in milliseconds.

The Stream Number provides a reference to identify which media streams
(in the ASF Data Unit's Stream Number field) are defined by a given
Stream Properties Object instance. Zero is an invalid stream number

(i.e., other Header Objects use stream number zero to refer to the
entire file as a whole rather than to a specific media stream within
the file).

The Stream Language ID Index field refers to the contents of the stream
itself (in other words, the language, if any, which the stream
uses/assumes).

Please see the Language List Object (Section 5.16) for the details
concerning how the Stream Language ID Index and Language ID Index
fields should be used.

The Stream Name Count field tells how many Stream Names are present.
Each stream name instance is potentially a localization into a specific
language. The Language ID Index field indicates the language in which
the Stream Name has been written in Unicode values.

The Stream Name Length field indicates the number of Unicode
"characters" that are found within the Stream Name field. The MIME Type
Length field indicates the number of bytes found within the MIME Type
field.

The Stream Name, MIME Type, and Stream Type are each mechanisms to
identify the Media Stream (in Unicode, MIME type, and GUID,
respectively).

The structure for the Type Specific Data field varies by media type.
The structure for this field for the Standard ASF Media Types is
detailed in Section 8.

5.3.1 Data Unit Extension Object

Mandatory: No
Quantity: 0 - n

The Data Unit Extension Object is an optional provision to include
application (or implementation)-specific data within each ASF Data Unit
(see Section 6.1) instance of this media stream.

Object Structure:

Field Name	Field Type	Size (bits)
Object ID	GUID	128
Object Size	UINT	64
Extension System	GUID	128
Data Unit Extension Size	UINT	16
Extension System Info Size	UINT	32
Extension System Info	UINT8	?

Notes:
Extension System is a GUID identifier of the type of information being
stored within the Extension Data field of the ASF Data Unit (see
Section 6.1).

The Data Unit Extension Size field indicates the number of bytes of
extension information that are present within the Extension Data field
of the ASF Data Unit (see Section 6.1) for this media stream. If the
Data Unit Extension Size field has a value of 0xFFFF (65535 decimal),
then the Extension Data field is variable length and the first byte of
the Extension Data field gives the length of the (following) extension
data for that particular ASF Data Unit instance. For example, if the
first byte of a variable sized entry has the value of "2," then two
additional extension data bytes will be present in that instance of the
Extension Data field.

The number, order, and size of the data elements within the ASF Data
Unit's Extension Data field directly correspond to the order in which
the Data Unit Extension Objects occur within the SPO for this media
stream. For example, assume that three Data Unit Extension Objects are
included within a stream's SPO. Assume that the first specifies a fixed
length of 4 bytes, the second specifies a variable length field, and
the third specifies a fixed length of 2 bytes. Therefore, the
Extension Data field of each ASF Data Unit for this stream will consist
of 4 bytes (extension #1), followed by 1 length byte plus up to 255
data bytes (extension #2), and finally 2 bytes (extension #3).

The Extension System Information field is an optional field providing
additional definitions or parameters (if any) of the Extension System.

5.4 Content Description Object

Mandatory: No
Quantity: 0 or 1

This object permits authors to record human-readable, pertinent data
about the file and its contents. This content is readily expandable to
satisfy varying bibliographic needs. Authors can supplement (or ignore)
the "standard" bibliographic information (for example, title, author,
copyright, and description) with content designations of their own
choosing. Each individual field name and value can be stored in as
many different languages as are preferred by the author, and can be
stream-specific or pertinent to the whole file.

Object Structure:

```
+------------------------+-------------+--------------+
|      Field Name:       | Field Type: | Size (bits): |
+------------------------+-------------+--------------+
| Object ID              |    GUID     |     128      |
| Object Size            |    UINT     |      64      |
| Description Record Count|   UINT     |      16      |
| Description Record     |  See below  |      ?       |
+------------------------+-------------+--------------+
```

Description Record:

```
+----------------------+---------------+------------+
|      Field Name:     |  Field Type:  | Size (bits):|
+----------------------+---------------+------------+
| Field Type           |     UINT      |     8      |
| Language ID Index     |UINT (see S5.16)|    16     |
| Stream Number         |     UINT      |    16      |
| Name Length           |     UINT      |    16      |
| Value Length          |     UINT      |    16      |
| Name                  |Unicode (UINT16)|    ?      |
| Value                 |Unicode (UINT16)|    ?      |
+----------------------+---------------+------------+
```

Notes:
The Object ID field contains the GUID for the Stream Properties Object
(see Appendix A). The Object Size is the length in bytes of this
object.

Description Record Count indicates the number of Description Records.

The Field Type field contains unsigned integer values.
* ISRC is the International Standard Recording Code as described in
  ISO 3901.
* ISWC is the International Standard Work code.
* UPC/EAN is the Universal Product Code/European Article Number (in
  other words, the "Bar code").
* Values 13 through 49 of the Field Types were derived from Reference
  [5]. The number in parentheses is the MARC tag value for that item.
* Values 50 through 60 of the Field Types were derived from Reference
  [6] for those elements that were not already obviously included
  within 8 through 45.
* Values 61 through 68 are RTCP SDES values and value 69 is the RTCP
  APP value. RTCP is defined within Reference [7]. Values 70 through
  73 are RTP header information that is also defined within Reference
  [7].

Please consult references [5], [6], and [7] for an interpretation of
the meanings of their field types.

The values of the Field Type field are:

1 = **Author**   2 = Title
3 = **Copyright**   4 = Description
5 = **Tool Name**   6 = Tool Version
7 =**Tool GUID**   8 = Date of Last Modification
9 = **Original Date Created**   10 = ISRC
11 = **ISWC**   12 = UPC/EAN
13 = **LCCN (10)**   14 = ISBN (20)
15 = **ISSN (22)**   16 = Cataloging Source, Leader (40)
17 = **Main Entry -- Personal Name (100)**
18 = **Main Entry - Corporate Name (110)**
19 = **Edition Statement (250)**   20 = Main Uniform Title (130)
21 = **Uniform Title (240)**   22 = Title Statement (245)
23 = **Varying Form Title (246)**
24 = **Publication, Distribution, and so on (260)**
25 = **Physical Description (300)**   26 = Added Entry Title (440)
27 = **Series Statement (490)**   28 = General Note (500)
29 = **Bibliography Note (504)**   30 = Contents Note (505)
31 = **Creation Credit (508)**   32 = Citation (510)
33 = **Participant (511)**   34 = Summary (520)
35 = **Target Audience (521)**   36 = Added Form Available (530)
37 = **System Details (538)**   38 = Awards (586)
39 = **Added Entry Personal Name (600)**   40 = Added Entry Topical Term (650)
41 = **Added Entry Geographic (651)**   42 = Index Term, Genre (655)
43 = **Tag Index Term, Curriculum (658) 44 = Added Entry Uniform Title (730)**
45 = **Added Entry Related (740)**
46 = **Series Statement Personal Name (800)**
47 = **Series Statement Uniform Title (830)**
48 = **Electronic Location and Access (856)**
49 = **Added Entry - Personal Name (700)** 50 = Coverage
51 = **Date**   52 = Resource Type
53 = **Format**   54 = Resource Identifier
55 = **Source**   56 = Language
57 = **Relation**   58 = Coverage
59 = **Subject**   60 = Contributor
61 = **CNAME**   62 = NAME
63 = **EMAIL**   64 = PHONE

65 = LOC   66 = TOOL
67 = NOTE   68 = PRIV
69 = APP   70 = SSRC
71 = Initial RTP Timestamp value   72 = Initial RTP Sequence Number
73= RTP Version Number

Values between 74 and 99 (inclusive) are reserved.
Values >= 100 are user-defined.

The Stream Number indicates whether the entry applies to a specific
media stream or whether it applies to the whole file. A value of zero
in this field indicates that it applies to the whole file; otherwise,
the entry applies only to the indicated stream number.

Name is in Unicode. This field may be blank if the Field Type value is
less than 100, unless the author explicitly wants to localize the name
of the field type.

The Name Length field indicates the number of Unicode "characters" that
are found within Name field. The Value Length field indicates the
number of Unicode "characters" that are found within Value field.

As a space optimization, a 16-bit Language ID Index field has been
used. See the Language List Object (Section 5.16) for more details.

5.5 Script Command Object

Mandatory: No
Quantity: 0 or 1

This object provides a list of Type/Parameter pairs of Unicode strings
that are synchronized to the ASF file's timeline. Types can include
"URL" or "FILENAME." These semantics and use of types are identical to
the Command Media Type (see Section 8.7). Other Type values may also be
freely defined and used. The semantics and treatment of this latter set
of Types are defined by the local implementations. The Parameter value
(referred to as "Commands" below) is specific to the type field. This
Type/Parameter pairing can be used for many purposes, including sending
URLs to be "launched" by a client into an HTML frame (in other words,
the "URL" type) or launching another ASF file for chained "continuous
play" audio or video presentations (in other words, the "FILENAME"
type). This object can also be used as an alternative method to stream
text (in addition to the Text Media Type) as well as to provide "script
commands" that can be used to control elements within the client
environment.

Object Structure:

```
+-------------------------+--------------+------------+
|      Field Name:        | Field Type:  | Size (bits):|
+-------------------------+--------------+------------+
| Object ID               |    GUID      |    128     |
| Object Size             |    UINT      |    64      |
| Type Count              |    UINT      |    16      |
| Command Count           |    UINT      |    16      |
| Types                   |  See below   |    ?       |
| Commands                |  See below   |    ?       |
+-------------------------+--------------+------------+
```

Types:

```
+----------------------+---------------+------------+
|      Field Name:     |  Field Type:  |Size (bits):|
+----------------------+---------------+------------+
| Type Name Length     |     UINT      |    16      |
| Type Name            |Unicode (UINT16)|    ?      |
+----------------------+---------------+------------+
```

Commands:

```
+----------------------+---------------+------------+
|      Field Name:     |  Field Type:  |Size (bits):|
+----------------------+---------------+------------+
| Presentation Time    |     UINT      |    32      |
| Type Index           |     UINT      |    16      |
| Command Name Length  |     UINT      |    16      |
| Command Name         |Unicode (UINT16)|    ?      |
+----------------------+---------------+------------+
```

Notes:
Presentation Time is given in millisecond granularities.

Types are stored as an array of Unicode strings, since they will
typically be reused. Commands specify their type using a zero-based
index into the array of Types.

The Type Name Length field indicates the number of Unicode "characters"
that are found within the Type Name field. The Command Name Length
field indicates the number of Unicode "characters" that are found
within the Command Name field.

5.6 Marker Object

Mandatory: No
Quantity: 0 or 1

This object contains a small, specialized index that is used to
provide named "jump points" within a file. This allows a content
author to divide a piece of content into logical sections such as song
boundaries in an entire CD or topic changes during a long presentation,
and to assign a human-readable name to each section of a file. This
index information is then available to the client to permit the user to
"jump" directly to those points within the presentation.

Object Structure:

Field Name:	Field Type:	Size (bits):
Object ID	GUID	128
Object Size	UINT	64
Index Specifier Count	UINT	16
Marker Count	UINT	16
Index Specifiers	See Section 5.14	?
Markers	See below	?

Markers:

Field Name:	Field Type:	Size (bits):
Presentation Time	UINT	32
Offsets	UINT64	?
Marker Name Count	UINT	16
Marker Names	See below	?

Marker Name:

Field Name:	Field Type:	Size (bits):
Language ID Index	UINT	16
Marker Name Length	UINT	16
Marker Name	Unicode (UINT16)	?

Notes:
The Index Specifiers are defined within the Index Parameters Object
(Section 5.14).

The Presentation Time is in millisecond granularities. This value does
not wrap around, which means that markers can only refer to the first
49.7 days of information contained within an ASF file.

Potentially multiple Offsets entries are listed within the Marker
structure. The number is determined by the requirement that there must
be one Offsets entry in each Marker structure for each Index Specifier
entry. Thus, the total size in bits of the Marker's Offsets field is
64 bits times the value of the Index Specifier Count field. An offset
value of 0xFFFFFFFFFFFFFFFF signifies that the entry contains an invalid
offset value.

As a space optimization, a 16-bit Language ID Index field has been
used. See the Language List Object (Section 5.16) for more details.

The Marker Name Length field indicates the number of Unicode
"characters" which are found within Marker Name field.

5.7 Component Download Object

Mandatory: No
Quantity: 0 or 1

This object provides a list of components (including version
information) required for the proper rendering of each stream in the
file. Each listed component has a human-readable name, a category
identifying the component type (which is usually either "codec" or
"renderer"), a component ID used to uniquely identify a specific
component, and version information for that component.

This object presupposes that the Component ID will be the primary
mechanism used to find the proper component to download. This object
purposefully does not use URLs to find these objects, for the following
reasons:

**1. Embedded URLs become stale very quickly, and end up being just
   wasted header space.**
**2. Legacy files and current components such as codecs have no knowledge
   of source URLs. Either authoring/conversion tools need to have**
   elaborate lookup tables so that they can embed the proper source
   URLs, or else the source URLs quickly become optional and,
   therefore, frequently omitted.
**3. Embedded source URLs would quickly become implementation-specific.
   Product A's authoring tools would embed pointers to product A's**
   playback components. When a Product B client got the source URL, it
   would have no way of knowing if it was talking to a general
   "component server" or a product-specific self-extracting download
   module.

Object Structure:

```
+-------------------------+--------------+------------+
|     Field Name:         |  Field Type: | Size (bits):|
+-------------------------+--------------+------------+
| Object ID               |     GUID     |    128     |
| Object Size             |     UINT     |    64      |
| Component Count         |     UINT     |    32      |
| Component Records       | See Below    |    ?       |
+-------------------------+--------------+------------+
```

Component Record:

```
+-------------------------+--------------+------------+
|     Field Name:         |  Field Type: | Size (bits):|
+-------------------------+--------------+------------+
| Category                |     GUID     |    128     |
| Component ID            |     GUID     |    128     |
| Version                 |     UINT     |    64      |
| Stream Number           |     UINT     |    16      |
| Component Name Length   |     UINT     |    16      |
| Component Name          |Unicode(UINT16)|    ?      |
+-------------------------+--------------+------------+
```

Notes:
The Component ID is a GUID that can use mappings for ACM and VCM
codecs, for example.

The Version field stores a "dotted quad" version stamp using the
highest 16 bits for the product version, the next 16 bits for the
incremental version, the next 16 bits for the revision, and the lowest
16 bits for the build number. The value 0.0.0.0 should be used for the
versions of ACM and VCM codecs. This value means "any version" and is
needed because there are no valid versioning numbers for ACM/VCM
codecs, since the "versioning information" is actually contained within
the Component ID's GUID value itself for these codec types. Other
entities that do not have valid version numbers should also use 0.0.0.0
in this field.

Stream Number identifies the multimedia stream associated with this
component. A 0 (zero) value means "all streams."

The Component Name is a human-readable display name for this component.

5.8 Stream Group Object

Mandatory: No
Quantity: 0 or 1

This object provides lists of "associated" streams that are grouped
into related presentation contexts. Each of these contexts contains a

Group Name by which these contexts may be referenced. This permits the
client to make implementation-specific composition and rendering

decisions affecting those streams. For associated image/video streams,
these decisions can include the number, size, and location of
image/video rendering windows, and their relative positions in three-
dimensional space. For audio streams, these decisions will impact the
potential mixing of associated audio streams that occur simultaneously
(stream start & end time can be determined using the Stream Properties
Object).

The following are additional examples of potential uses of this object:
1. **A file containing two video streams (such as a TV newscast with a
   large image of the anchorperson and a smaller image of the news**
   story) would have each video stream in a separate group. A client
   implementation could then use external compositional information to
   decide that the video stream containing the news story (whose
   natural size is known in the Stream Properties Object's type-
   specific data field) should be superimposed in the top-right corner
   of the larger anchorperson video stream.
2. **A file containing multi-track audio would group all of those audio
   streams together (perhaps along with associated video and lyrics for**
   a karaoke effect). This might tell the client implementation that
   these streams should be mixed.
3. **A file containing two separate image streams (for example, JPEGs,
   and GIFs) could group the streams together. This might tell the**
   client to "mix" them, by logically rendering them into the same
   window. Another approach would be to make two different groups,
   which would imply that images from the two streams could be visible
   at the same time.

The default behavior if no Stream Group Object is present within the
File Header (and therefore no stream groups are defined) is to assume
that all streams are grouped together.

Object Structure:

List of stream groupings, each of which contains a list of stream
numbers for that grouping. Each stream grouping is optionally assigned
a Group Name that can serve as a "handle" by which the group as a whole
may be referenced. This name may be localized into different languages.

```
+-------------------------+---------------+-------------+
|     Field Name:         |  Field Type:  | Size (bits):|
+-------------------------+---------------+-------------+
| Object ID               |     GUID      |    128      |
| Object Size             |     UINT      |     64      |
| Stream Group Count      |     UINT      |     16      |
| Stream Groups           |   See Below   |     ?       |
+-------------------------+---------------+-------------+
```

Stream Group:
```
+-------------------------+--------------+------------+
|     Field Name:         | Field Type:  | Size (bits):|
+-------------------------+--------------+------------+
| Group Name Count        |    UINT      |    16      |
| Group Names             |  See Below   |    ?       |
| Stream Count            |    UINT      |    16      |
| Stream Numbers          |   UINT16     |    ?       |
+-------------------------+--------------+------------+
```

Group Name:
```
+-------------------------+--------------+------------+
|     Field Name:         | Field Type:  | Size (bits):|
+-------------------------+--------------+------------+
| Language ID Index       |    UINT      |    16      |
| Group Name Length       |    UINT      |    16      |
| Group Name              |Unicode(UINT16)|   ?       |
+-------------------------+--------------+------------+
```

Notes:
See the Language List Object (Section 5.16) for more details concerning
how to use the Language ID Index field.

Media streams, which have been grouped into Group Names-named logical
units, are grouped by enumerating their stream numbers in the Stream
Numbers field. The Stream Count field identifies how many media streams
are enumerated within the Stream Numbers field.

The Group Name Length field indicates the number of Unicode
"characters" that are found within Group Name field.

5.9 Scalable Object

Mandatory: No
Quantity: 0 - n

This object stores the dependency relationships between all of the
media streams that comprise logical bands of the same scalable media.
It can be used for scalable audio and video, as well as other types of
scalable streams. Along with the dependency relationships among the
streams, this object stores a default sequence in which the streams
should be used when implementations are doing dynamic bandwidth
scaling.

Object Structure:
The object consists of a list of Dependency Info "structures" for each
stream that comprises a logical band of the same scalable stream.

A Dependency Info "structure" (in other words, the Dependency Record)
contains:
1. **Stream Number.**
2. **List of stream numbers upon which this stream depends.**

The object also contains an author-determined default sequence (in
other words, the Default Sequence Record) that indicates the
preferential order in which the streams should be used (in other words,
items listed first should, by default, be used first). Each entry in
this list consists of the following two fields:
1. **Stream Number**
2. **Enhancement GUID.**

```
+-------------------------+--------------+------------+
|     Field Name:         |  Field Type: | Size (bits):|
+-------------------------+--------------+------------+
| Object ID               |     GUID     |    128     |
| Object Size             |     UINT     |     64     |
| Record Count            |     UINT     |     16     |
| Default Sequence Records|   See Below  |     ?      |
| Dependency Records      |   See Below  |     ?      |
+-------------------------+--------------+------------+
```

Default Sequence Record:
```
+-------------------------+--------------+------------+
|     Field Name:         |  Field Type: | Size (bits):|
+-------------------------+--------------+------------+
| Stream Number           |     UINT     |     16     |
| Enhancement Type        |     GUID     |    128     |
+-------------------------+--------------+------------+
```

Dependency Record:
```
+-------------------------+--------------+------------+
|     Field Name:         |  Field Type: | Size (bits):|
+-------------------------+--------------+------------+
| Stream Number           |     UINT     |     16     |
| Dependent Stream Count  |     UINT     |     16     |
| Dependent Stream Numbers|     UINT16   |     ?      |
+-------------------------+--------------+------------+
```

Notes:
The Record Count field stores both the number of Default Sequence
Records and the number of Dependency Records (in other words, the same
number of each). This number is equivalent to the number of streams
involved in this scaleability relationship.

Possible Enhancement GUID Values are None, Unknown, Temporal, Spatial,
Quality, Stereo (Audio), and Frequency Response (Audio).

5.10 **Prioritization Object**

Mandatory: No
Quantity: 0 or 1

This object indicates the author's intentions as to which streams
should or should not be dropped in response to varying network
congestion situations. There may be special cases where this
preferential order may be ignored (for example, the user hits the
"mute" button). However, generally it is expected that implementations
will try to honor the author's preference.

Priority determinations are made solely with reference to base streams
(in other words, this includes non-scalable streams and the base layer
only of scalable streams). The author can indicate their preference as
to what should happen to enhancement layer streams by means of the
bandwidth restriction field.

The priority of each stream is indicated by how early in the list that
stream's stream number is listed (in other words, the list is ordered
in terms of decreasing priority). Two additional fields provide
associated information:

1) The "Mandatory/Optional" field identifies whether the author wants
   that stream kept "regardless" (in other words, the Mandatory bit is
   set) or whether they are willing to have that stream dropped (in
   other words, an optional stream that is indicated by the Mandatory
   bit being cleared). Optional streams must never be assigned a higher
   priority than mandatory streams.
2) The Bandwidth Restriction field permits the author to indicate how
   much of the available bandwidth will be used. For example, if the
   stream is a base layer of a scalable codec, the bandwidth will
   determine how many enhancement layers may be selected. This number
   is determined by the dependency relationships and priority ordering
   information found within the Scalable Object combined with the
   bandwidth information contained within each stream's Stream
   Properties Object.

Streams that are in a mutual exclusion relationship with each other (for
example, languages) should all be listed in adjacent order (in other
words, priority n, n+1, n+2, and so on), sorted in decreasing order of
maximum stream bandwidth. When bandwidth calculations are made, only
the bandwidth used by the selected stream in a mutual exclusion
relationship will be computed; each non-selected stream in such a
relationship will be ignored. This combination of prioritization and
mutual exclusion can be used to create scalable content even though

scalable codecs have not been used by means of creating multiple
distinct media stream instances of the "same content," each at
different bandwidths.

Object Structure:
```
+------------------------+--------------+------------+
|     Field Name:        | Field Type:  | Size (bits):|
+------------------------+--------------+------------+
| Object ID              |    GUID      |    128     |
| Object Size            |    UINT      |    64      |
| Priority Record Count  |    UINT      |    16      |
| Priority Records       |  See Below   |    ?       |
+------------------------+--------------+------------+
```

Priority Record:
```
+------------------------+--------------+------------+
|     Field Name:        | Field Type:  | Size (bits):|
+------------------------+--------------+------------+
| Stream Number          |    UINT      |    16      |
| Priority Flags         |    UINT      |    16      |
 ----+                   +--------------+------------+
     | Mandatory         |              | 1 (LSB)    |
     | Reserved          |              |    15      |
+----+-------------------+--------------+------------+
| Bandwidth Restriction  |    UINT      |    32      |
+------------------------+--------------+------------+
```

Notes:
Priority Records are listed in order of decreasing priority.

The Stream Number should only specify the base stream (if it is
scalable).

Bandwidth Restriction is in bits per second. A value of 0 (zero)
indicates "no restriction."

5.11 Mutual Exclusion Object

Mandatory: No
Quantity: 0 - n

This object identifies streams that have a mutual exclusion
relationship to each other (in other words, only one of the streams
within such a relationship can be streamed - the rest are ignored).
There should be one instance of this object for each set of objects
that contain a mutual exclusion relationship. The exclusion type is

used so that implementations can allow user selection of common
choices, such as language.

Object Structure:

```
+------------------------+--------------+------------+
|    Field Name:         |  Field Type: | Size (bits):|
+------------------------+--------------+------------+
| Object ID              |     GUID     |    128     |
| Object Size            |     UINT     |     64     |
| Exclusion Type         |     GUID     |    128     |
| Stream Number Count    |     UINT     |     16     |
| Stream Numbers         |    UINT16    |     ?      |
+------------------------+--------------+------------+
```

Notes:
The Exclusion Type identifies the nature of that mutual exclusion
relationship (for example, language).

The Stream Number Count indicates how many Stream Numbers are in the
Stream Numbers list. Each of the media streams in this list is in a
mutual exclusion relationship with the others.

5.12 Inter-Media Dependency Object

Mandatory: No
Quantity: 0 or 1

This object provides the capability for an author to identify
dependencies between different media types. An example of such a
relationship would be to specify that a video effects stream will be
presented only if a certain enhancement layer of a video codec is also
currently being presented. Another example is binding a timecode media
stream to another media stream to provide alternate timecodes for that
other stream's data.

Object Structure:

List of Dependency Info "structures" for any stream involved in an
inter-media dependency relationship.

```
+------------------------+--------------+------------+
|    Field Name:         |  Field Type: | Size (bits):|
+------------------------+--------------+------------+
| Object ID              |     GUID     |    128     |
| Object Size            |     UINT     |     64     |
| Dependency Record Count |    UINT     |     16     |
| Dependency Records     |See Section 5.9|    ?      |
+------------------------+--------------+------------+
```

Notes:
The Dependency Record structure is given in Section 5.9.

The Dependency Record Count indicates the number of Dependency Records
present.

Should multiple dependencies be listed within the Dependent Stream
Numbers fields of a single Dependency Record, these dependencies are in
a Boolean AND relationship to each other (in other words, the stream
number is dependent upon x AND y). Boolean OR relationships (in other
words, the stream number is dependent upon x OR y) are indicated by
having multiple Dependency Record entries, each having the same Stream
Number value in the Stream Number field of the Dependency Record.
Streams that are dependent upon either one stream or another, or
optionally both, are said to be in an OR dependency relationship.

5.13 Rating Object

Mandatory: No
Quantity: 0 or 1

This object contains W3C-defined Platform for Internet Content
Selection (PICS) information (see references [1] and [2]). PICS
establishes Internet conventions for label formats. It thus provides a
basis for specifying the rating of the multimedia content within an ASF
file. This object does not specify the specific rating service that is
to be used. The content creator is consequently able to use the rating
service of their choice, as long as it is specified according to the
PICS conventions.

Object Structure:
```
+-------------------------+--------------+------------+
|     Field Name:         |  Field Type: | Size (bits):|
+-------------------------+--------------+------------+
| Object ID               |     GUID     |    128     |
| Object Size             |     UINT     |    64      |
| PICS Data               |     UINT8    |    ?       |
+-------------------------+--------------+------------+
```

Note:
PICS information is stored as opaque data in an RFC 822-conformant
format (see reference [3]).

5.14 Index Parameters Object

Mandatory: Yes if index is present in file; Otherwise no.
Quantity: 0 or 1

This object supplies a sufficient amount of information to regenerate
the index for an ASF file should the original index have been omitted
or deleted. It includes only information about those streams that are
actually indexed (there must be at least one stream in an index).

Object Structure:
```
+------------------------+--------------+------------+
|     Field Name:        | Field Type:  | Size (bits):|
+------------------------+--------------+------------+
| Object ID              |    GUID      |    128     |
| Object Size            |    UINT      |    64      |
|Index Entry Time Interval|   UINT      |    32      |
| Index Specifier Count  |    UINT      |    16      |
| Index Specifiers       |  See Below   |    ?       |
+------------------------+--------------+------------+
```

Index Specifier:
```
+------------------------+--------------+------------+
|     Field Name:        | Field Type:  | Size (bits):|
+------------------------+--------------+------------+
| Stream Number          |    UINT      |    16      |
| Index Type             |    UINT      |    16      |
+------------------------+--------------+------------+
```

Notes:
The Index Entry Time Interval is in milliseconds.

The Index Specifier Count field identifies how many Index Specifier
entries exist within the Index Specifiers field.

Every Index Type requires all index entry offsets to be to a data unit
boundary of an ASF Data Unit containing data for the specified Stream
Number. Also, the send time of that data unit must not exceed the time
of the index entry, which is a presentation time.

Index Type values are as follows: 1 = Nearest Data Unit, 2 = Nearest
Object, and 3 = Nearest Clean Point. The Nearest Data Unit indexes
point to the data unit the send time of which is closest to the index
entry time. The Nearest Object indexes point to the closest data unit
containing an entire object or first fragment of an object. The Nearest
Clean Point indexes point to the closest data unit containing an entire
object (or first fragment of an object) that has the Clean Point Flag
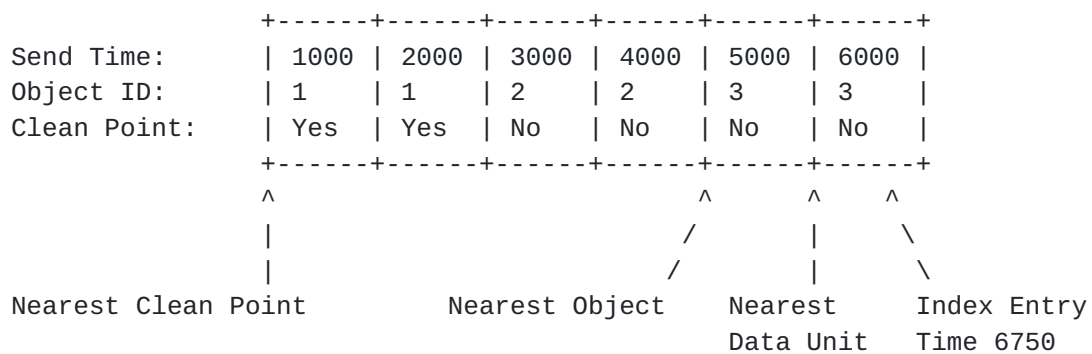set.

```
              +------+------+------+------+------+------+
Send Time:    | 1000 | 2000 | 3000 | 4000 | 5000 | 6000 |
Object ID:    | 1    | 1    | 2    | 2    | 3    | 3    |
Clean Point:  | Yes  | Yes  | No   | No   | No   | No   |
              +------+------+------+------+------+------+
               ^                         ^      ^      ^
               |                        /       |       \
               |                       /        |        \
Nearest Clean Point        Nearest Object    Nearest    Index Entry
                                             Data Unit   Time 6750
                 Figure 3: Explanation of Indexing Terms
```

5.15 Color Table Object

Mandatory: No
Quantity: 0 to n

This object contains a color table that is used by one or more media
streams. For purposes of reference, each color table is given a unique
identifier for reference purposes.

Object Structure:

Field Name:	Field Type:	Size (bits):
Object ID	GUID	128
Object Size	UINT	64
Color Table ID	GUID	128
Color Table Record Count	UINT	16
Color Table Record	See Below	?

Color Table Record:

Field Name:	Field Type:	Size (bits):
Red	UINT	8
Green	UINT	8
Blue	UINT	8

Note:
The structure consists of a list of Color Table Records, which contain
RGB triplets.

5.16 Language List Object

```
Mandatory:      Yes
Quantity:       1
```

This object contains an array of ASCII-based Language IDs. All other
header objects refer to languages through zero-based positions into
this array.

Object Structure:
```
+-------------------------+---------------+-------------+
|     Field Name:         |  Field Type:  | Size (bits):|
+-------------------------+---------------+-------------+
| Object ID               |      GUID     |     128     |
| Object Size             |      UINT     |      64     |
| Language ID Count       |      UINT     |      16     |
| Language ID Records     |    See Below  |      ?      |
+-------------------------+---------------+-------------+
```

Language ID Record:
```
+-------------------------+---------------+-------------+
|     Field Name:         |  Field Type:  | Size (bits):|
+-------------------------+---------------+-------------+
| Language ID Length      |      UINT     |      8      |
| Language ID             |  ASCII (UNIT8)|      ?      |
+-------------------------+---------------+-------------+
```

Notes:
Other objects refer to the Language List Object by means of their own
Language List ID Index fields. The value within the Language ID Index
field explicitly provides an index into the Language ID Record
structure in order to identify a specific language. The first entry
into this structure has an index value of 0 (zero). Index values that
are greater than the number of entries within the Language ID Record
structure are interpreted as signifying "American English."

The Language ID Length field indicates the size in bytes of the
Language ID field.

[6](#) **Data Object**

Mandatory: Yes
Quantity: 1

This object contains all of the ASF Data Units for a file. These data
units are organized in terms of increasing send times. Each ASF Data Unit
contains data from only a single media stream. This data may consist of
an entire object from that stream. Alternatively, it can consist of a
partial object of that stream (fragmentation) or several concatenated
objects from that stream (grouping).

The structure of the data object contains the following two fields,
which are immediately followed by one or more instances of ASF Data
Units.

```
+-------------------------+--------------+-------------+
|     Field Name:         |  Field Type: | Size (bits):|
+-------------------------+--------------+-------------+
| Object ID               |     GUID     |    128      |
| Object Size             |     UINT     |     64      |
+-------------------------+--------------+-------------+
```

6.1 ASF Data Unit Definition

In general, ASF media types logically consist of sub-elements that are
referred to as objects. What an object happens to be in a given media
stream is entirely media stream-dependent (for example, it is a
specific image within an image media stream, a frame within a (non-
scalable) video stream, etc). It is efficient to try to fit a media
stream's object into a single ASF Data Unit whenever possible. When
that is not possible, we can fragment the object (if it is too big) or
group the object (if it is too little) with other objects within that
same media stream when forming a data unit. In any case, each ASF Data
Unit is a conveniently sized grouping of data from a single media type.

ASF data units have the following format:
```
+-------------------------+--------------+-------------+
|     Field Name:         |  Field Type: | Size (bits):|
+-------------------------+--------------+-------------+
| Data Unit Length        |     UINT     | 16 or 32    |
| Stream Number           |     UINT     |     16      |
| Send Time               |     UINT     |     32      |
| Data Unit Flags         |     UINT     |  8 or 32    |
+----+                    +--------------+-------------+
     | Extended Flags     |              | 1 (LSB)     |
     | Clean Point        |              |     1       |
     | Fragment           |              |     1       |
     | Fragment Size      |              |     1       |
     | Grouped Data       |              |     1       |
     | Reserved           |              |     3       |
+----+--------------------+--------------+-------------+
| Object Number           |     UINT     |     8       |
| Presentation Time       |     UINT     | 0, 16, or 32|
| Offset Into Object      |     UINT     | 0, 16, or 32|
| Object Size             |     UINT     | 0, 16, or 32|
| Extension Data          |     UINT8    |     ?       |
| Data Unit Data          |     UINT8    |     ?       |
+-------------------------+--------------+-------------+
```

Notes:
The Data Unit Length Field specifies the length in bytes of that ASF
Data Unit. The Huge Data Units Flag (in the Flags field of the File
Properties Object) determines the size of the Data Unit Length field.
In general, it is strongly recommended that the 16-bit size alternative
of the Data Unit Length field should be used and that the maximum size
value for this field should not exceed 65,000. All ASF Data Units must
be smaller (in bytes) than the value indicated by the Maximum Data Unit
Size field within the File Properties Object. Thus, the value of the
Data Unit Length field can never exceed the Maximum Data Unit Size
value.

The Stream Number identifies the media stream data of which is
contained within the Data Unit Data field of this ASF Data Unit. The
value of the Stream Number field corresponds to the Stream Number value
within this media stream's Stream Properties Object.

The Send Time is in milliseconds and refers to the intrinsic timeline
of the ASF file (which begins at value 0). The value of this field
"wraps around" to zero every 2**32 milliseconds (which is roughly every
49.7 days).

The following give the significance of the Data Unit Flags:
* The size of the Data Unit Flags field is determined by whether the
  Extended Flags flag is set or cleared. If it is cleared, then there
  are only 8 bits of flags present. If it is set, then there are 32
  bits of flags with the value of the highest order 3 bytes being
  reserved.
* The Clean Point Flag identifies whether this data unit is a "clean
  point" (for example, video key frame) or not.
* The Fragment Flag indicates whether this data unit contains a
  fragment of an object or not. If the Fragment Flag is set, then the
  Offset Into Object and Object Size fields exist within this ASF Data
  Unit instance. These fields are used to indicate the breakup of
  large object across data unit boundaries. If this flag is cleared,
  then these two fields do not exist within this ASF Data Unit
  instance. If the Fragment Flag is set, then the Grouped Data Flag
  must be cleared. If an object containing a clean point is
  fragmented, the Clean Point Flag is set for all of the fragments
  of that object.
* The Fragment Size flag is valid only if the Fragment Flag has been
  set. If the Fragment Size Flag is cleared, then the Offset Into

   Object and Object Size fields are 16 bits long. If it is set, then
   these fields are 32-bits long.
* The Grouped Data Flag indicates whether or not multiple objects from
   the same stream are grouped together into a single data unit. The
   Grouped Data flag must be cleared (in other words, indicating no
   grouped data) if the Fragment Flag is set. Grouping consists of
   prefixing a 16-bit length field to the object data. A 16-bit delta
   time (in milliseconds) is inserted between each length-object
   pairing. For example:

```
                    +----------------+
                    |  16-bit Length  |
                    +----------------+
                    |      Data       |
                    +----------------+ ----------
                    |16-bit Delta Time|
                    +----------------+    Repeat
                    |  16-bit Length  |   1 - N
                    +----------------+    Times
                    |      Data       |
                    +----------------+ ----------
                    Figure 4: Grouping
```
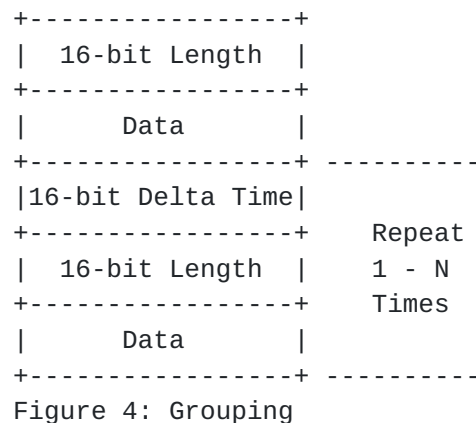
The 16-bit Delta Time field is always included within Grouped Data
as shown above. This field indicates a presentation time for the
following grouped object. If the Presentation Time flags within the
Stream Properties Object are configured to state that presentation
times are not used (value of 00), then the value of the 16-bit Delta
Time field of the Grouped Data indicates the difference in send
times between the two objects. In this case, the delta time
effectively indicates a presentation time difference for the grouped
objects only.

Should an object containing a clean point be grouped, the object
containing the clean point must be the first object in the grouping.
All other objects in a grouping are interpreted as not being clean
points.

The Object Number field identifies which object within the data stream
is being sent. (The first object is Object Number 0.) The value of this
field "wraps" around to 0 every 2**8 objects. It should be explicitly
noted that the term "object" within the context of ASF media types (and
hence the Object Number field of the ASF Data Unit) is entirely
unrelated to the ASF Object definition, which was given in Section 3.1.

The Presentation Time Flags within the Stream Properties Object
determine whether the Presentation Time field exists or not. Those
flags also determine whether the Presentation time is full presentation
time (in other words, full 32-bit reference to the timeline) or whether
the presentation time is a 16-bit delta off of the send time. All
presentation times are in terms of the Rational Unit values established
for that media stream within the Stream Properties Object.

The Offset Into Object and Object Size fields are used exclusively for
fragmentation. The former identifies the offset into the object
(identified by the Object Number field) where the current fragment
begins, and the Object Size identifies the total size of that object.
These fields provide the information needed to reconstruct the object
when it is received at the client.

The Extension Data field is optional and its existence and size is
determined by the optional presence of one or more Data Unit Extension
Object(s) (see Section 5.3.1) within the Stream Properties Object (see
Section 5.3). The Extension System (GUID) field within the Data Unit
Extension Object(s) establishes the semantics of the Extension Data.

6.2 ASF Data Unit Examples

The following examples are provided to help explain how the data unit
format may appear in various usage scenarios. In each case excerpts
from the example Stream Properties Object must be included, since they
determine the actual data unit composition. Also, it is assumed in all
examples that the Huge Data Units Flag within the File Properties
Object has been cleared.

6.2.1 Complete Key Frame Example:

The Presentation Time Flags in the Stream Properties Object specify
that the Presentation Delta is in the data units (in other words, value
"10"). The Extension Data Size value (of the Data Unit Extension
Object) is 2.

The following is an example data unit for the case where the Object
Number is 5, the Send Time is 5000, and the Presentation Time is 5750:

Field Name:	Field Size (bytes):	Field Value:
Data Unit Length	2	1014
Stream Number	2	1
Send Time	4	5000
Data Unit Flags	1	0x02
Object Number	1	5
Presentation Time	2	750

```
| Extension Data          | 2                 | Opaque      |
| Data Unit Data          | 1000              | Opaque      |
+-------------------------+-------------------+-------------+
```

6.2.2 Partial JPEG Example:

The Presentation Time Flags in the Stream Properties Object specify
that presentation times are not used (value "00"). The Extension Data
Size value (of the Data Unit Extension Object) is 0.

The following is an example data unit for the case where bytes 1000
through 1799 are being sent for a 4000-byte-long JPEG image at a Send
Time of 7000. The Object Number of this JPEG image is 17.

Field Name:	Field Size (bytes):	Field Value:
Data Unit Length	2	814
Stream Number	2	2
Send Time	4	7000
Data Unit Flags	1	0x06
Object Number	1	17
Offset Into Object	2	1000
Object Size	2	4000
Data Unit Data	800	Opaque

6.2.3 Three Delta Frames Example

The Presentation Time Flags in the Stream Properties Object specifies
that the Presentation Time Delta is carried in the data units (value
"10"). The Extension Data Size value (of the Data Unit Extension
Object) is 0.

The following is an example of a data unit containing three delta video
frames. The first is 20 bytes long, and presents at 8500, the second
is 30 bytes long and presents at 8533, and the third is 40 bytes long
and presents at 8575.

Field Name:	Field Size (Bytes):	Field Value:
Data Unit Length	2	112
Stream Number	2	1
Send Time	4	8000

```
| Data Unit Flags                            | 1      | 0x10        |
| Object Number                              | 1      | 97          |
| Presentation Time                          | 2      | 500         |
| Data Unit Data                             | 100    | See Below   |
+-----|                 +------------------+---------+-------------+
      | [Object ID #97] | Data Length      | 2       | 20          |
      |                 | Data             | 20      | Opaque      |
      |-----------------+------------------+---------+-------------+
      | [Object ID #98] | Pres. Time Delta | 2       | 33          |
      |                 | Data Length      | 2       | 30          |
      |                 | Data             | 30      | Opaque      |
      |-----------------+------------------+---------+-------------+
      | [Object ID #99] | Pres. Time Delta | 2       | 42          |
      |                 | Data Length      | 2       | 40          |
      |                 | Data             | 40      | Opaque      |
+--------------------------------------------+---------+-------------+
```

[Note concerning the example above: 8533 minus 8500 forms the
Presentation Time Delta value of 33 for Object ID #98. 8575 minus 8533
forms the Presentation Time Delta value of 42 for Object ID #99.]

[7](#) Index Object

Mandatory: No, but strongly recommended
Quantity: 0 or 1

This top-level ASF object supplies the necessary indexing information
for an ASF file. It includes stream-specific indexing information
based on an adjustable index entry time interval. The index is
designed to be broken into blocks to facilitate storage that is more
space-efficient by using 32-bit offsets relative to a 64-bit base. That
is, each index block has a full 64-bit offset in the block header,
which is added to the 32-bit offsets found in each index entry. If a
file is larger than 2^32 bytes, then multiple index blocks can be used
to fully index the entire large file while still keeping index entry
offsets at 32 bits.

Indices into the Index Object are in terms of Presentation Times. The
corresponding Offset field values (of the Index Entry, see below) are
byte offsets that, when combined with the Index Block's Block Position
value, indicate the starting location of an ASF Data Unit.

The Index Object is not recommended to be used for files where the Send
Time of the first Data Unit within the Data Object has a Send Time value
significantly greater than zero (otherwise the index itself will be sparse
and inefficient). In such cases, an offset value of 0xFFFFFFFF is used to
indicate an invalid offset value. Invalid offsets signify that this
particular index entry does not identify a valid indexable point.

Invalid offsets may occur for the initial index entries of a media
stream whose first ASF Data Unit has a non-zero send time.

Fleischman   [Page 43]

Object Structure:
```
+------------------------+---------------+------------+
|     Field Name:        |  Field Type:  | Size (bits):|
+------------------------+---------------+------------+
| Object ID              |      GUID     |     128    |
| Object Size            |      UINT     |      64    |
|Index Entry Time Interval|     UINT     |      32    |
| Index Specifier Count  |      UINT     |      16    |
| Index Specifiers       |See Section 5.14|     ?     |
| Index Block Count      |      UINT     |      32    |
| Index Blocks           |   See Below   |      ?     |
+------------------------+---------------+------------+
```

Index Block:
```
+------------------------+---------------+------------+
|     Field Name:        |  Field Type:  | Size (bits):|
+------------------------+---------------+------------+
| Block Position         |      UINT     |      64    |
| Index Entry Count      |      UINT     |      32    |
| Index Entries          |   See Below   |      ?     |
+------------------------+---------------+------------+
```

Index Entry:
```
+------------------------+---------------+------------+
|     Field Name:        |  Field Type:  | Size (bits):|
+------------------------+---------------+------------+
| Offsets                |     UINT32    |      ?     |
+------------------------+---------------+------------+
```

Notes:
Block Position is the byte offset of the beginning of this block
relative to the beginning of the first Data Unit (i.e., the
beginning of the Data Object + 24 bytes).

Index Entry Count is the number of Index Entries in the block.

The size of the Offsets field within each Index Entry structure is
(32 bits multiplied by the value of the Index Specifier Count
field). For example, if the Index Specifier Count is 3, then there
are three 32-bit offsets in each Index Entry. Index Entry offsets
are ordered according to the ordering specified by the Index
Parameters Object, thereby permitting the same stream to be
potentially indexed by multiple Index Types (e.g., Nearest Clean
Point, Nearest Object, Nearest Data Unit).

The Index Entry Time Interval has a millisecond granularity. [Note: the
problem with making index entries be based upon rational time units is
that each stream can have its own choice of rational time units - which

would make selecting the one to be used for index problematical.]

8 **Standard ASF Media Types**

ASF files store a wide variety of multimedia content. It is natural to
expect implementations to make use of this content to produce rich
multimedia experiences. It is anticipated that implementations will
flexibly produce unique media types of their own creation. It is
highly desirable, however, that a rich set of standard media types be
commonly supported to permit content compatibility between diverse
implementations.

The purpose of this section is to define a set of Standard ASF Media
Types. [Note: "Media types", as used in this document, is roughly
equivalent to the IETF RFC 1590 term "content type."] The explicit

intention of this section is that if an implementation supports a media
type defined within this section (in other words, audio, video, image,
timecode, text, MIDI, command, Media Object), that media type must be
supported in the manner described within this section if the
implementation is to be considered to be "content-compliant" with the
ASF specification. This commonality will hopefully define a minimum
subset of media within which multi-vendor interoperability will be
possible. This, in turn, will simplify media exchange between
companies, developers, and individuals. No restrictions are placed upon
how implementations support non-standard media types (in other words,
media types other than those covered in this section).

There are two elements to each Media Type definition:
1. **Identification of the information that will populate the Type-
   Specific Data field of the Stream Properties Object. This provides**
   media-specific information needed to interpret the data in the media
   stream.
2. **Description of the media stream data itself.**
   Each of the following sub-sections will define the core media types in
   terms of these two elements.

8.1   Audio Media Type

Type-Specific Data:

+---------------------------+---------------+-------------+
| Field Name: | Field Type: | Size (bits):|
+---------------------------+---------------+-------------+
Codec ID	GUID	128
Error Concealment Type	GUID	128
Bits per Sample	UINT	32
Samples per Second	UINT	32
Average Frame Size	UINT	32
Maximum Frame Size	UINT	32
Samples per Frame	UINT	32
Flags	UINT	16
	Reserved		16
Number of Channels	UINT	16
Error Concealment Data Size	UINT	16
Codec Specific Data Size	UINT	16
Error Concealment Data	UINT8	?
Codec Specific Data	UINT8	?
+---------------------------+---------------+-------------+

Media Stream Format:
Output of a codec or sampling device.

Notes:
The Bits per Sample field should have a value of 0 (zero) if a variable
bit-rate compression scheme is used.

The term "frame" in this context refers to the compressed chunk of data
produced by an audio codec.

8.1.1 Scrambled Audio

One Error Concealment Type is so-called "scrambled audio." This refers
to an error concealment approach that mitigates the impact of lost
audio data units by rearranging the order in which audio data is sent.
The Scrambled Audio concealment scheme stores audio data in a
rearranged fashion on disk. This disk order is maintained as the data
is streamed over a network. The client must correctly unscramble the
audio data before submitting it to the codec to decompress. This
approach works well for fixed bit-rate audio codecs that have no inter-
frame dependencies.

The Error Concealment Data field has the following structure for this
approach:

```
+---------------------------+---------------+-------------+
|     Field Name:           | Field Type:   | Size (bits):|
+---------------------------+---------------+-------------+
| Audio Object Size         |     UINT      |     32      |
| Rearranged Chunk Size     |     UINT      |     32      |
| Chunks per Data Unit      |     UINT      |     32      |
| Chunk Distance            |     UINT      |     32      |
+---------------------------+---------------+-------------+
```

Notes:
The Audio Object Size refers to the size in bytes of all rearranged
audio objects in this stream. Other object sizes are possible but will
not use this concealment scheme.

Rearranged Chunk Size refers to the size in bytes of audio blocks that
are rearranged within each object. This value should be a multiple of
the Average Frame Size.

Chunks per Data Unit refers to the number of Rearranged Chunk Size
audio blocks that are contained in each ASF data unit for this stream.

Chunk Distance refers to the number of audio chunks to skip when
filling data units.

Every data unit except for the one containing the "end" of each audio
object will always contain (Chunks per Data Unit) * (Rearranged Chunk
Size) bytes of audio.

The following diagram illustrates how audio scrambling will be done.

Original Audio Media "chunks" before scrambling:

```
        +-----+-----+-----+-----+-----+-----+-----+
        |  1  |  2  |  3  |  4  |  5  |  6  |  7  |
        +-----+-----+-----+-----+-----+-----+-----+
```

Each rectangle represents the Rearranged Chunk Size.
The size of all rectangles added together represents the Audio Object
Size.
If it is configured so that the Chunk Distance = 2 and the Chunks per
Packet = 2, the following would be the resulting packet order as stored
on the disk (and streamed across the network):

```
        +-----+     +-----+     +-----+     +-----+
        |  1  |     |  7  |     |  5  |     |  6  |
        |  4  |     |  2  |     |  3  |     |     |
        +-----+     +-----+     +-----+     +-----+
```

[8.2](#) Video Media Type

Type-Specific Data:

Field Name:	Field Type:	Size (bits):
Codec ID	GUID	128
Color Table ID	GUID	128
Average Frame Rate	FLOAT	64
Average Key Frame Rate	FLOAT	64
Maximum Key Frame Rate	FLOAT	64
Average Frame Size	UINT	32
Maximum Frame Size	UINT	32
Flags	UINT	16
| Reserved		16
Encoded Image Width	UINT	16
Encoded Image Height	UINT	16
Display Image Width	UINT	16
Display Image Height	UINT	16
Color Depth	UINT	16
Codec Specific Data Size	UINT	16
Codec Specific Data	UINT8	?

Media Stream Format:
Output of a codec or sampling device.

Notes:
The Encoded/Display Image Width/Height is in pixels.

The Average Key Frame Rate and the Maximum Key Frame Rate are able to
indicate very slow rates as a fractional value. For example, a frame
rate of one frame every 8 seconds would be shown as 0.125.

Key Frames are also known as Clean Points within the ASF Data Unit (see
Section 6.1). Key Frames are known as I-Frames in MPEG terminology.

8.3 **Image Media Type**

Type-Specific Data:

```
+---------------------------+---------------+-------------+
|     Field Name:           |  Field Type:  | Size (bits):|
+---------------------------+---------------+-------------+
| Codec ID                  |     GUID      |    128      |
| Color Table ID            |     GUID      |    128      |
| Maximum Image Size        |     UINT      |    32       |
| Encoded Image Width       |     UINT      |    16       |
| Encoded Image Height      |     UINT      |    16       |
| Display Image Width       |     UINT      |    16       |
| Display Image Height      |     UINT      |    16       |
| Flags                     |     UINT      |    16       |
|      | Reserved           |               |    16       |
| Color Depth               |     UINT      |    16       |
| Codec Specific Data Size  |     UINT      |    16       |
| Codec Specific Data       |     UINT8     |    ?        |
+---------------------------+---------------+-------------+
```

Media Stream Format:
The data contents of one or more logical Image files.

Notes:
The following Image Types must be supported on all ASF clients: Loss-
Tolerant JPEG and JPEG. Other Image Types may also be optionally
supported. [Note: Loss-Tolerant JPEG is a Microsoft-defined JPEG
variant that will be described in a future version of this document.]

The Codec ID will include GUIDs for many image formats, including Loss-Tolerant JPEG, GIF, and JPEG.

The Color Table ID is used to indicate the palette when Color Depth is
8 **bpp.**

The Encoded/Display Image Width/Height is in pixels.

The Maximum Image Size is specified in bytes.

The existence, content, and size of Codec Specific Data is keyed off of the Codec ID.

8.4 Timecode Media Type

Type-Specific Data:

Field Name:	Field Type:	Size (bits):
Timecode ID	GUID	128

Media Stream Format:
Timecodes of the type indicated by the Timecode ID.

Notes:
The Timecode ID will contain GUIDs for SMPTE.

It is expected that a timecode media stream will be bound to specific other media streams by means of the Inter-Media Dependency object. This will provide a basis for establishing (non-mathematic) SMPTE timecode for that media stream (in other words, Rational Presentation Times solely are able to establish mathematically based timecodes). For example, if an SMPTE timecode is bound to a video stream, entries with the same send times in the two streams are paired, thereby permitting SMPTE timecodes to be given to that video stream.

8.5 Text Media Type

Type-Specific Data:

Field Name:	Field Type:	Size (bits):
Text Encoding System	GUID	128
Encoding Specific Data	??	??

Media Stream Format:
Text Media shall be streamed as NULL-terminated streams.

Notes:
The following Text Types must be supported on all ASF clients: ASCII,
Unicode, and HTML. Other Text Types may also be optionally supported.

The Encoding Specific Data field will have a different meaning
depending on the Text type identified within the Text ID field:
* If ASCII or Unicode is the Text Encoding System, then the
  Encoding Specific Data field will not exist.
* If HTML, then this may optionally contain a Cascading Style Sheet
  (CSS) that will be in common across each of the HTML objects
  within this media stream.

All ASF implementations are required to support ASCII and are strongly
encouraged to support Unicode and HTML.

As is the case with the other media types, all rendering and
composition decisions for Text Media (for example, overlays, Z-
ordering, positioning, marquis, and so on) are made by out-of-band
techniques alluded to in Section 5.8.

Should "text files" be streamed, each "file" is considered to be an
object within this data stream (in other words, it will have a distinct
Object ID value within the ASF Data Unit (see Section 6.1)).

8.6 MIDI Media Type

The goals for the definition of the MIDI media type were to incur
minimal overhead for MIDI data while maintaining extensibility for
future enhancements. Also, it was desirable to enable reasonable
granularity seeking operations within MIDI streams. We believe that
this proposal meets the stated objectives.

Minimal overhead is present in the definition of the MIDI event
structure (see the Media Stream Format section below). Usually, only
two bytes more than MIDI's standard overhead is required, while
maintaining a more accurate timing model.

Extensibility is built in through an event class system, which permits
the mapping and assignment of globally unique identifiers (GUIDs) to
the integer-based event classes contained in a MIDI stream.

Seeking operations are supported through an expanded use of the Clean
Point concept. On some interval throughout a seekable MIDI stream,

objects will need to begin with what is termed "Clean Point Info"
events. These events will serve to re-establish the state of patch
changes and controllers at that point in the MIDI stream. Those
objects that contain this Clean Point Info can then be marked using the
Clean Point Flag in the ASF data unit definition, and indexed using the
normal ASF Index. During the course of normal streaming playback,
these redundant Clean Point Info events are ignored. When seeking, the
client uses these events to re-establish the current state of patches
and controllers. An exact list of which controllers' state should be
preserved is TBD.

Type-Specific Data:

Field Name:	Field Type:	Size (bits):
Flags	UINT	16
| Extended Classes		1 (LSB)
| Extended Channels		1
| Reserved		14
Event Class Count	UINT	16
Event Classes	GUID	?

Notes:
The Extended Classes Flag means that every MIDI event in this stream
uses the 8 bit Extended Event Class field (see below) to extend the
number of possible event classes from 63 to 16383 (by extending the
event class space from 6 bits to 14 bits).

The Extended Channels Flag means that every MIDI event in this stream
is followed by a byte that contains an additional 8 bits of MIDI
channel information, permitting the use of 4096 channels instead of
just the traditional 16 channels.

The Event Classes list of GUIDs contains the mapping used for this
particular stream from the GUID identifiers for MIDI event classes to
the integers used in this stream. The first entry in this list is
given the integer value 1 (one), since 0 (zero) is reserved to indicate
a standard MIDI event.

It is expected that MIDI streams will have the Reliable Flag set in
their Stream Properties Object, as the loss of MIDI data generally
leads to undesirable and unpredictable results.

Media Stream Format:

Each object within a MIDI stream will contain an array of the following
MIDI Event structures:

```
+---------------------------+---------------+-------------+
|     Field Name:           | Field Type:   | Size (bits):|
+---------------------------+---------------+-------------+
| Presentation Time Delta   |     UINT      |     16      |
|                           |     UINT      |      8      |
|    | Event Size Present   |               |    1 (LSB)  |
|    | Clean Point Event    |               |      1      |
|    | Event Class          |               |      6      |
| Extended Event Class      |     UINT      |    0 or 8   |
| Event Size                |     UINT      |    0 or 32  |
| MIDI Event                |     UINT8     |      ?      |
| Extended Channel Info     |     UINT      |    0 or 8   |
+---------------------------+---------------+-------------+
```

Notes:
The Presentation Time Delta field is stored in units of 100
microseconds (tenths of milliseconds). The 16-bit size of the field,
when combined with the chosen time units, permits ASF MIDI objects to
contain up to 6.5535 seconds worth of MIDI data in a single object.
The delta is based on the explicit or implicit Presentation Time value
of the object in the ASF MIDI stream. Each event stores an individual
time delta from the base presentation time of the object (for ease of
manipulation), so the resulting presentation time for every single MIDI
event in the same object can be computed as object presentation time +
presentation time delta. All MIDI events in a single object must be
stored in sorted order of increasing presentation time deltas.

The Event Size Present field is used to indicate that an explicit 32-
bit event size field is being used in this particular event. This will
typically be useful for SYSEX events whose lengths can not be
predicted. If not present, the size of the MIDI Event field must be
implicitly determined based on the event's contents. In the case of a
standard MIDI event (with Event Class == 0), a simple table can be used
to map from MIDI status byte values to the overall size of the MIDI
event data. Recall that if the stream's Extended Channels flag is set,
then an Extended Channel Info byte follows the standard MIDI event.

The Clean Point Event field indicates that this particular MIDI event
should only be processed if received immediately following a seek
operation. Otherwise, client implementations should skip this event.

The Event Class field is used as a 1-based index into the Event Classes
list of GUIDs stored in the stream header. Event Class 0 (zero) is
reserved to indicate a Standard MIDI event. The Extended Event Class
field is used to expand the number of simultaneously permissible event
classes for a particular stream from 63 to 16383 by extending the
number of event class bits from 6 to 14. It occurs only if the

Extended Classes flag is set in the stream header.

The Event Size field is used only if the Event Size Present field is
set, as was previously mentioned.

MIDI running status can be used between the events contained within one
individual ASF object (or buffer), but should not cross object
boundaries. This recommendation is designed to simplify client
playback resource requirements and implementations.

8.7 Command Media Type

Type-Specific Data:
```
+--------------------------+---------------+-------------+
|      Field Name          |  Field Type   | Size (bits) |
+--------------------------+---------------+-------------+
| Command Type             |     GUID      |    128      |
+--------------------------+---------------+-------------+
```

Media Stream Format:
The data of URL Command Types complies with the URL format strings as
defined in RFC 1738 and RFC 2017. These strings shall be NULL
terminated ASCII strings. Frame values are indicated by a "&" delimiter
according to the following syntax: "& frame & URL \0".

The data of the FILENAME Command Type either complies with the URL
Command Type format or else the format used on the local operating
system to indicate ASCII filenames.

Notes:
There are two standard Command Type GUIDs: URL and FILENAME. The URL
command indicates that the URL is to be "launched" by a client into an
HTML window or frame. The FILENAME command indicates the ASF file
indicated is to be played immediately (for example, for "continuous
play" environments).

It is required that all ASF implementations support fully specified
URLs for both URL and FILENAME uses. Relative path URLs may be
optionally supported. The use of Local URLs (in other words, those
containing O/S dependent references such as drive letters) is
discouraged but not prohibited.

8.8 Media-Objects (Hotspot) Media Type

The goal of the Media-Objects stream is to encode an object
representation of a related visual media stream (for example, video,
image, slideshow, animation, and so on) and the interactive features
associated with these objects. This is accomplished by "binding" the

media object stream to the related visual media stream by means of the
Inter-Media Dependency Object.

Theoretically, the Media Object stream will enable elements within the
visual media stream to be referred to in an object-oriented fashion (in
addition to the traditional image-oriented fashion). This approach
enhances the information level embedded in a visual media stream,
providing both the developer and the viewer with a new, more natural
method of referencing the logical objects in the media. For example,
derived applications may include object-based interactivity, object-
based storage and retrieval and object-based statistics.

Type-Specific Data:

Field Name:	Field Type:	Size (bits)	Description:
Horizontal Resolution	UINT	16	The horizontal resolution of the frame. This parameter is used to interpret the object's geometry parameters.
Vertical Resolution	UINT	16	The vertical resolution of the frame. This parameter is used to interpret the object's geometry parameters.
Number of Commands	UINT	16	Total number of Command Entries.
Command Entry Array	Command Entry Structure	??	

Command Entry Structure:

Field Name:	Field Type:	Size (bits)	Description:
Link Type	OBLinkType	8	The command type which will be activated when actuating the object.
Link Command	See Below		

For URL Command:

URL	URL Format	??	The full URL address.

```
|           | String      |      | Identical to the URL Command |
|           |             |      | type (see Section 8.7).      |
+-----------+-------------+------+------------------------------+
```

Seek to Time Command:

+------------+-------------+------+-----------------------------+
Time	Timestamp	32	The point in time within the
			stream to seek to. This value
			has a millisecond granularity.
+------------+-------------+------+-----------------------------+

 Seek to Marker Command:

+------------+-------------+------+-----------------------------+
Marker	UINT	32	The point in the stream to
			seek to (in reference to
			index locations indicated by
			the Marker Object (Section
			5.6)). Values exceeding the
			number of marker object
			indexes will be ignored.
+------------+-------------+------+-----------------------------+

 For Filename Command:

+------------+-------------+------+-----------------------------+
| Filename | String | ?? | Identical to the Filename |
| | | | Command (see Section 8.7). |
+------------+-------------+------+-----------------------------+

 For Script Command:

+------------+-------------+------+-----------------------------+
| Type Field | UINT | 8 | Number of Unicode characters |
| Size | | | in the Type Field. |
+------------+-------------+------+-----------------------------+
| Value Field| UINT | 16 | Number of Unicode characters |
| Size | | | in the Value Field. |
+------------+-------------+------+-----------------------------+
| Type Field | Unicode | ?? | The Type field (e.g., Script |
| | | | Name). |
+------------+-------------+------+-----------------------------+
| Value Field| Unicode | ?? | The Value field (e.g., |
| | | | Script Contents). |
+------------+-------------+------+-----------------------------+
No Link Command field is present for Pause, Resume, Exit, and
Same-Value commands.
+----------------+-------------+------+-----------------------------+

Notes:
The Horizontal and Vertical Resolution parameters determine the units
by which the objects' geometry will be defined. These parameters
describe the number of "logical units" in each frame width and height.

This relative representation provides easy interface for objects' re-sizing and media scaling.

The Link Type defines the command that is linked to the object. This command is activated by a mouse-click upon the object. OBLinkType defines one of the following commands:
0 = NO_LINK (nothing happens upon mouse click)
1 = URL (flip a URL page)
2 = SeekToTime
3 = SeekToMarker
4 = Filename (jump to another ASF file)
5 = Script (Type/Value pair whose actual meaning (semantics) is locally
defined. For example, the Type may indicate a script name and the Value may indicate the contents of the script body.)
6 = Pause
7 = Resume (ignore if pause had not previously been hit)
8 = Exit
9 = Same-Value: Continue to use the command which had been previously
specified for this Object ID. [Note: if there was not a previously specified command for this Object ID, then the command for this Object ID will default to NO_LINK. This command type should not be used for instances in which the Command Entry Structure has been appended to the Object Structure of the Media Object Stream.]
Values greater than 9 are Reserved

The Marker Object mentioned for the Seek to Marker command is defined in Section 5.6.

Media Stream Format:
The following describes the structure of each object instance. Multiple object instances can optionally be directly concatenated together as an array of structures in one ASF Data Unit. Every instance encodes the object description and/or interactive features for a given duration. Each description is valid from its Start Time until its End Time.

Field Name:	Field Type:	Size (bits)	Description:
Object ID	UINT	16	A unique identifier of the object
Start Time	UINT	32	The starting time of this instance of the object (presentation time value)
End Time	UINT	32	The ending time of this instance of the object (presentation time value)

+-----------------+-------------+------+-----------------------------+
| Object Shape | OBShape | 4 | The primitive shape of the |
| | | | hotspot |
+-----------------+-------------+------+-----------------------------+
Object Flags	OBFlags	4 -low	Different Flags assigned to
		order	the object (could be used by
		nibble	any external application)
+-----------------+-------------+------+-----------------------------+
Object Geometry	See Below	(4*16)	
		or (N*	
		2*16)	
+----+   +-------------+------+-----------------------------+

For primitive shape objects (Rectangle, Triangle, Ellipse, etc.):

Left	UINT	16	X coordinate of the top-left corner of the bounding rectangle
Top	UINT	16	Y coordinate of the top-left corner of the bounding rectangle
Right	UINT	16	X coordinate of the bottom-right corner of the bounding rectangle
Bottom	UINT	16	Y coordinate of the bottom-right corner of the bounding rectangle

For Polygon shape object:

X1	UINT	16	X coordinate of the first vertex of the polygon
Y1	UINT	16	Y coordinate of the first vertex of the polygon
Xn ...	UINT	16	X coordinate of the n-th vertex of the polygon
Yn ...	UINT	16	Y coordinate of the n-th vertex of the polygon
XN	UINT	16	X coordinate of the last vertex of the polygon
YN	UINT	16	Y coordinate of the last vertex of the polygon

```
+-----------------+-------------+------+-----------------------------+
| Effects Field   | UINT        |  8   | Cursor and visual effects   |
+----+            +-------------+------+-----------------------------+
     | Cursor Type| OBCursor    |  4   | Cursor effects              |
     +------------+-------------+------+-----------------------------+
     | Marking    | OBMark      |4 -low| Marking effects             |
     | Type       |             |nibble|                             |
+----+------------+-------------+------+-----------------------------+
| Index           | UINT        | 16   | The command which will be   |
|                 |             |      | activated when actuating    |
|                 |             |      | this object                 |
+-----------------+-------------+------+-----------------------------+
```

Notes:
Object ID is a unique identifier of the object, throughout its life
span.

The Start Time and End Time parameters are interpreted according the
presentation time granularities of the visual media stream to which
this particular Media Object stream was bound by means of the Inter-
media Dependency Object.

Object Shape selects one of the pre-defined shapes: 0 = Rectangle, 1 =
Triangle, 2 = Ellipse, and 4 = Polygon.

Object Flags field is defined in an implementation-specific manner. The
default value of this field is zero. Clients may optionally ignore this
field.

The object geometry parameters are all represented in the
Horizontal/Vertical Resolution units, which are defined in the stream
header.

For all primitive shapes (in other words Rectangle, Ellipse, Triangle),
defining the bounding rectangle of the shape is sufficient to fully
describe the shape. (That is also true, for an isosceles triangle with
a horizontal base. For any other type of triangle, the polygon shape
can be used.)

The Cursor Type specifies the author's preference for cursor shape.
OBCursor values are:
0 = arrow
1 = hand
2 = hide cursor
3 - 10 Implementation Specific
11 - 15 Reserved

Implementations may use the Implementation specific values in an
implementation-specific manner. Clients may also optionally ignore
interpreting the Cursor Type field altogether at their own discretion.

The Marker Type visual effects associated with a hot spot. OBMark
values are:
0 **= none**
1 **= invert**
2 **= darken**
3 **= outline**
4 **- 10 Implementation Specific**
11 **- 15 Reserved**

Fleischman   [Page 58]

Implementations may use the Implementation specific values in an
implementation-specific manner. Clients may also optionally ignore
interpreting the Cursor Type field altogether at their own discretion.

The Index value refers to which entry in the Command List Array (within
the Stream Properties Object) is being activated. Index values
exceeding the number of entries within the Command List Array will be
ignored unless it is 0xFFFF (in other words, 65535 decimal). A value of
0xFFFF signifies that a Command Entry Structure is appended to this
object structure instance (for example, to support Real-Time Editing).

Acknowledgements

The Advanced Streaming Format (ASF) Specification was co-authored by
Microsoft Corporation, RealNetworks, Intel Corporation, Adobe Systems
Incorporated, and Vivo Software, Inc. Microsoft owns the copyright
for the ASF Specification and is responsible for publishing the ASF
Specification and any modifications thereto.

In 1996, Microsoft developed a preliminary version of ASF and
implemented it within its NetShow (tm) streaming server and client
products. Microsoft Corporation, RealNetworks, Intel Corporation,
Adobe Systems Incorporated, and Vivo Software, Inc then enhanced
this preliminary version and authored an initial "straw man" draft
of the ASF Specification.

A first draft version of the ASF Specification was generated based upon
the comments and feedback of an additional 45 companies. Following this,
the ASF specification was made available for public comment, in which
roughly 100 corporations and universities participated. On September
30, 1997, Microsoft announced the free public availability of the
completed specification. Several versions of the ASF Specification
containing errata (e.g., clarifications) have subsequently been published
at http://www.microsoft.com/asf/specs.htm. This document reflects the
latest version of the ASF Specification (i.e., February 1998).

Microsoft Intellectual Property Statement

royalty-free license under applicable Microsoft rights to implement
and use the technology described in this contribution. One condition
of this license shall be the party's agreement not to assert patent
rights against Microsoft and other companies for their implementation
of the contribution. Microsoft expressly reserves all other rights
it may have in the material and subject matter of this contribution.
Microsoft expressly disclaims any and all warranties regarding this
contribution including any warranty that (a) this contribution does
not violate the rights of others, (b) the owners, if any, of other
rights in this contribution have been informed of the rights and
permissions granted to ISOC herein or (c) any required authorizations
from such owners have been obtained.

Submitter's Address

Eric Fleischman
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399 United States
Electronic mail: ericfl@microsoft.com

Bibliography

[1]   T. Krauskopf, J. Miller, P. Resnick, and G. W. Treese, "Label
        Syntax and Communication Protocols," World Wide Web Consortium
        http://www.w3.org/PICS/labels.html, May 5 1996.

[2]   J. Miller, P. Resnick, and D. Singer, "Rating Services and Rating
        Systems (and Their Machine Readable Descriptions)," World Wide Web
        Consortium http://www.w3.org/PICS/services.html, May 5 1996.

[3]   D. Crocker, "RFC 822: Standard for the Format of ARPA Internet
        Text Messages," ftp://ds.internic.net/rfc/rfc822.txt, August 1982.

[4]   H. Alvestrand, "RFC 1766: Tags for the Identification of
        Languages," ftp://ds.internic.net/rfc/rfc1766.txt, March 2, 1995.

[5]   "MARC Bibliographic Formats,"
        http://www.fsc.follett.com/data/marctags/.

[6]   "Dublin Core Elements," ftp://ds.internic.net/internet-
        drafts/draft-kunze-dc-01.txt or
        http://purl.org/metadata/dublin_core_elements/.

[7]   H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RFC 1889:
        RTP: A Transport Protocol for Real-Time Applications," January 1996;
        ftp://ds.internic.net/rfc/rfc1889.txt.

Appendix A: ASF GUIDs

Use of GUIDs within ASF
GUIDs are used to uniquely identify all objects and entities within ASF
files. This provides the foundation for the extensibility and
flexibility that characterizes ASF. For example, versioning is
transparently supported within ASF by this mechanism. That is, since
each version of an ASF object has its own unique GUID, the ASF library
knows how to interpret the semantics and syntax of any given version of
that object based upon the GUID that is used.

Similarly, each ASF multimedia object type is uniquely identified by a

GUID. New media types can be created, identified by their own GUID, and inserted into ASF data streams.

Similarly, new codec types, new error correction approaches, and novel
innovations of all types can be readily invented, identified by GUIDs
and used within ASF.

New ASF object types (for example, see Other Objects as is shown in
Figure 2 of Section 3.2 as well as explicit text within Sections 5.1
and 5.3) may be defined. This forms a chief "extensibility feature" of
ASF to support new innovations and inventions as they arise. Each new
ASF object type needs its own unique GUID identification.

ASF GUIDs
The following are standard GUIDs that have been defined for all ASF
objects and GUID-based fields within this specification. This list is
not exhaustive. Implementations may supplement this list with
additional GUIDs when necessary to identify entities/elements/ideas
that have not yet been enumerated by this appendix.

Microsoft will endeavor to maintain a list of the additional GUID
definitions (about which it has been informed) at a public Web site.
The initial location of this web site will be
http://www.microsoft.com/asf/ Companies desiring to register additional
GUID definitions should send an email message to ASF@microsoft.com.

Standard Base ASF Objects GUIDs
ASF Header Object   {D6E229D1-35DA-11d1-9034-00A0C90349BE}
ASF Data Object   {D6E229D2-35DA-11d1-9034-00A0C90349BE}
ASF Index Object   {D6E229D3-35DA-11d1-9034-00A0C90349BE}

Standard ASF Header Object GUIDs
File Properties Object   {D6E229D0-35DA-11d1-9034-00A0C90349BE}
Stream Properties Object   {D6E229D4-35DA-11d1-9034-00A0C90349BE}
Data Unit Extension Object   {D6E22A0F-35DA-11d1-9034-00A0C90349BE}
Content Description Object   {D6E229D5-35DA-11d1-9034-00A0C90349BE}
Script Command Object   {D6E229D6-35DA-11d1-9034-00A0C90349BE}
Marker Object   {D6E229D7-35DA-11d1-9034-00A0C90349BE}
Component Download Object   {D6E229D8-35DA-11d1-9034-00A0C90349BE}
Stream Group Object   {D6E229D9-35DA-11d1-9034-00A0C90349BE}
Scalable Object   {D6E229DA-35DA-11d1-9034-00A0C90349BE}
Prioritization Object   {D6E229DB-35DA-11d1-9034-00A0C90349BE}
Mutual Exclusion Object   {D6E229DC-35DA-11d1-9034-00A0C90349BE}
Inter-Media Dependency Object {D6E229DD-35DA-11d1-9034-00A0C90349BE}
Rating Object   {D6E229DE-35DA-11d1-9034-00A0C90349BE}
Index Parameters Object   {D6E229DF-35DA-11d1-9034-00A0C90349BE}
Color Table Object   {D6E229E0-35DA-11d1-9034-00A0C90349BE}
Language List Object   {D6E229E1-35DA-11d1-9034-00A0C90349BE}

Other ASF Header Object GUIDs
ASF Placeholder Object   {D6E22A0E-35DA-11d1-9034-00A0C90349BE}

Standard GUIDs for the Stream Type Field of the Stream Properties
Object
Audio Media   {D6E229E2-35DA-11d1-9034-00A0C90349BE}
Video Media   {D6E229E3-35DA-11d1-9034-00A0C90349BE}
Image Media   {D6E229E4-35DA-11d1-9034-00A0C90349BE}
Timecode Media   {D6E229E5-35DA-11d1-9034-00A0C90349BE}
Text Media   {D6E229E6-35DA-11d1-9034-00A0C90349BE}
MIDI Media   {D6E229E7-35DA-11d1-9034-00A0C90349BE}
Command Media   {D6E229E8-35DA-11d1-9034-00A0C90349BE}
Media-Object (Hotspot) {D6E229FF-35DA-11d1-9034-00A0C90349BE}

Codecs for Audio and Video Media Types
A GUID is needed for each version of a codec implementation that
produces dissimilar encodings of the same input. Microsoft will
maintain a list of GUIDs according to their Codec/version number at a
Microsoft Web site. The initial location of this site is
http://www.microsoft.com/asf/ Companies that want to register the GUIDs
of additional Codec/version numbers should send their registrations to
ASF@microsoft.com.

GUIDs for the Error Concealment Type Field of the Audio Media Type
No Error Concealment   {D6E229EA-35DA-11d1-9034-00A0C90349BE}
Scrambled Audio (see Section 8.1.1) {D6E229EB-35DA-11d1-9034-
00A0C90349BE}

GUIDs for the Color Table ID field of the Video and Image Media Types
No Color Table   {D6E229EC-35DA-11d1-9034-00A0C90349BE}

GUIDs for the Timecode ID of the Timecode Media Type
SMPTE Time   {D6E229ED-35DA-11d1-9034-00A0C90349BE}

GUIDs for the Text Encoding System Field of the Text Media Type
ASCII Text   {D6E229EE-35DA-11d1-9034-00A0C90349BE}
Unicode Text   {D6E229EF-35DA-11d1-9034-00A0C90349BE}
HTML Text   {D6E229F0-35DA-11d1-9034-00A0C90349BE}

GUIDs for the Extension System Field of the Data Unit Extension Object
RTP Extension Data   {96800c63-4c94-11d1-837b-0080c7a37f95}

GUIDs for the Command Type Field of the Command Media Type
URL Command   {D6E229F1-35DA-11d1-9034-00A0C90349BE}
Filename Command   {D6E229F2-35DA-11d1-9034-00A0C90349BE}

GUIDs for the Category Field of the Component Download Object
ACM Codec   {D6E229F3-35DA-11d1-9034-00A0C90349BE}

VCM Codec   {D6E229F4-35DA-11d1-9034-00A0C90349BE}
QuickTime Codec   {D6E229F5-35DA-11d1-9034-00A0C90349BE}
DirectShow Transform Filter {D6E229F6-35DA-11d1-9034-00A0C90349BE}
DirectShow Rendering Filter {D6E229F7-35DA-11d1-9034-00A0C90349BE}

Enhancement GUIDs for the Scalable Object
No Enhancement   {D6E229F8-35DA-11d1-9034-00A0C90349BE}
Unknown Enhancement Type   {D6E229F9-35DA-11d1-9034-00A0C90349BE}
Temporal Enhancement   {D6E229FA-35DA-11d1-9034-00A0C90349BE}
Spatial Enhancement   {D6E229FB-35DA-11d1-9034-00A0C90349BE}
Quality Enhancement   {D6E229FC-35DA-11d1-9034-00A0C90349BE}
Number of Channels Enhancement (for example, Stereo)
                             {D6E229FD-35DA-11d1-9034-00A0C90349BE}
Frequency Response Enhancement {D6E229FE-35DA-11d1-9034-00A0C90349BE}

GUIDs for the Exclusion Type Field of the Mutual Exclusion Object
Language   {D6E22A00-35DA-11d1-9034-00A0C90349BE}
Same Content at Different Bit Rates
                             {D6E22A01-35DA-11d1-9034-00A0C90349BE}
Unknown Reason   {D6E22A02-35DA-11d1-9034-00A0C90349BE}

Appendix B: Bit Stream Types

The bit stream type describes the target data type and the order of
transmission of bits in the coded bit stream. The bit stream types are
ASCII, GUID, FILETIME, UINT, and Unicode.

ASCII:
A UINT8 (see UINT below) value containing ASCII data. ASCII data is
defined in RFC 1766.

FILETIME:
A 64-bit integer that contains a time stamp corresponding to the number
of 100 nanosecond ticks since January 1, 1601. The following diagram
demonstrates the filetime format:

```
(MSB)                                                           (LSB)
+--------+--------+--------+--------+--------+--------+--------+--------+
| byte 0 | byte 1 | byte 2 | byte 3 | byte 4 | byte 5 | byte 6 | byte 7 |
+--------+--------+--------+--------+--------+--------+--------+--------+
<---------------------------- 64 bits ---------------------------->
```

The GMT time zone is used for all filetime entries.

GUID:
The terms GUID (globally unique identifier) and UUID (universally
unique identifier) are identical. GUIDs are a 128-bit (16 octet) data
structure composed of a 32-bit unsigned integer, two 16-bit unsigned
integers, and an array of eight octets. The constituent parts are shown
in the following diagrams:

```
(MSB)                          (LSB)
+-------+-------+-------+-------+
|byte 0 |byte 1 |byte 2 |byte 3 |
+-------+-------+-------+-------+
<------------32 bits------------>
UNSIGNED INTEGER


(MSB)        (LSB)
+-------+-------+
|byte 0 |byte 1 |
+-------+-------+
<----16 bits---->
UNSIGNED INTEGER


(MSB)        (LSB)
+-------+-------+
|byte 0 |byte 1 |
+-------+-------+
<----16 bits---->
UNSIGNED INTEGER


(MSB)                          (LSB)
+-------+-------+...+-------+-------+
|byte 0 |byte 1 |...|byte 7 |byte 8 |
+-------+-------+...+-------+-------+
<--------------64 bits------------->|
FIXED-LENGTH ARRAY
```

These components are concatenated to form the UUID:

```
(MSB)                                                        (LSB)
+-------+-------+-------+-------+-------+-------+...+-------+-------+
|byte 0 |byte 1 |byte 2 |byte 3 |byte 4 |byte 5 |...|byte 14|byte 15|
+-------+-------+-------+-------+-------+-------+...+-------+-------+
<---------------------------128 bits--------------------------->
UNIVERSALLY UNIQUE IDENTIFIER (UUID)
```

UINT:
Unsigned integer in Little-Endian byte and Little-Endian bit order.
When a number is appended to UINT, the number refers to the number of
bits contained within this unsigned integer value. For example:
* UINT64 is an unsigned integer value that is 64 bits long
* UINT32 is an unsigned integer value that is 32 bits long
* UINT16 is an unsigned integer value that is 16 bits long
* UINT8 is an unsigned integer value that is 8 bits long.

UNICODE:
A UINT16 (see UINT above) value containing Unicode data.

Appendix C: GUIDs and UUIDs

ABSTRACT

This appendix describes the format of UUIDs (Universally Unique
IDentifier), which are also known as GUIDs (Globally Unique
IDentifier). A GUID is 128 bits long, and if generated according to the
one of the mechanisms in this document, is either guaranteed to be
different from all other UUIDs/GUIDs generated until 3400 A.D. or
extremely likely to be different (depending on the mechanism chosen).
GUIDs were originally used in the Network Computing System (NCS) [1]
and later in the Open Software Foundation's (OSF) Distributed Computing
Environment [2].

This specification is derived from the latter specification with the
kind permission of the OSF.

Introduction

This specification defines the format of UUIDs (Universally Unique
IDentifiers), also known as GUIDs (Globally Unique IDentifiers). A GUID
is 128 bits long, and if generated according to the one of the
mechanisms in this document, is either guaranteed to be different from
all other UUIDs/GUIDs generated until 3400 A.D. or extremely likely to
be different (depending on the mechanism chosen).

Motivation

One of the main reasons for using GUIDs is that no centralized
authority is required to administer them (beyond the one that allocates
IEEE 802.1 node identifiers). As a result, generation on demand can be
completely automated, and they can be used for a wide variety of
purposes. The GUID generation algorithm described here supports very

high allocation rates: 10 million per second per machine if you need
it, so that they could even be used as transaction IDs. GUIDs are
fixed-size (128 bits), which is reasonably small relative to other
alternatives. This fixed, relatively small size lends itself well to
sorting, ordering, hashing of all sorts, storing in databases, simple
allocation, and ease of programming in general.
Specification

A GUID is an identifier that is unique across both space and time, with
respect to the space of all GUIDs. To be precise, the GUID consists of
a finite bit space. Thus the time value used for constructing a GUID is
limited and will roll over in the future (at approximately A.D. 3400,
based on the specified algorithm). A GUID can be used for multiple
purposes, from tagging objects with an extremely short lifetime, to
reliably identifying very persistent objects across a network.

The generation of GUIDs does not require that a registration authority
be contacted for each identifier. Instead, it requires a unique value
over space for each GUID generator. This spatially unique value is
specified as an IEEE 802 address, which is usually already available to
network-connected systems. This 48-bit address can be assigned based on
an address block obtained through the IEEE registration authority. This
section of the GUID specification assumes the availability of an IEEE
802 address to a system desiring to generate a GUID, but if one is not
available, Section 4 specifies a way to generate a probabilistically
unique one that can not conflict with any properly assigned IEEE 802
address.

C.1   Format

The following table gives the format of a GUID.

Field:	Data Type	Octet #	Note:
time_low	UINT32	0 - 3	The low field of the timestamp
time_mid	UINT16	4 - 5	The middle field of the timestamp
time_hi_and_version	UINT16	6 - 7	The high field of the timestamp multiplexed with the version number
clock_seq_hi_and_res-erved	UINT8	8	The high field of the clock sequence multiplexed with the variant
time_low	UINT8	9	The low field of the clock sequence
node	UINT8 array	10-15	The spatially unique node identifier

The GUID consists of a record of 16 octets and must not contain padding
between fields. The total size is 128 bits.

To minimize confusion about bit assignments within octets, the GUID
record definition is defined only in terms of fields that are integral
numbers of octets. The version number is multiplexed with the timestamp
(time_high), and the variant field is multiplexed with the clock
sequence (clock_seq_high).

The timestamp is a 60-bit value. For GUID version 1, this is
represented by Coordinated Universal Time (UTC) as a count of 100-
nanosecond intervals since 00:00:00.00, 15 October 1582 (the date of
Gregorian reform to the Christian calendar).

The version number is multiplexed in the 4 most significant bits of the
time_hi_and_version field.

The following table lists currently defined versions of the GUID.

msb1	msb2	msb3	msb4	Version	Description
0	0	0	1	1	DCE version
0	0	3	0	2	DCE security version with
					embedded POSIX UIDs

The variant field determines the layout of the GUID. The structure of
DCE GUIDs is fixed across different versions. Other GUID variants may
not interoperate with DCE GUIDs. Interoperability of GUIDs is defined
as the applicability of operations such as string conversion,
comparison, and lexical ordering across different systems. The variant
field consists of a variable number of the MSBS of the
clock_seq_hi_and_reserved field.

The following table lists the contents of the DCE variant field.

msb1	msb2	msb3	Description:
0	-	-	Reserved, NCS backward compatibility
1	0	-	DCE variant
1	1	0	Reserved, Microsoft Corporation GUID
1	1	1	Reserved for future definition

The clock sequence is required to detect potential losses of
monotonicity of the clock. Thus, this value marks discontinuities and

prevents duplicates. An algorithm for generating this value is outlined in the "Clock Sequence" section below.

The clock sequence is encoded in the 6 least significant bits of the clock_seq_hi_and_reserved field and in the clock_seq_low field.

The node field consists of the IEEE address, which is usually the host address. For systems with multiple IEEE 802 nodes, any available node address can be used. The lowest addressed octet (octet number 10) contains the global/local bit and the unicast/multicast bit, and is the first octet of the address transmitted on an 802.3 LAN.

Depending on the network data representation, the multi-octet unsigned integer fields are subject to byte swapping when communicated between different endian machines.

The nil GUID is special form of GUID that is specified to have all 128 bits set to 0 (zero).

C.2   Algorithms for Creating a GUID

Various aspects of the algorithm for creating a GUID are discussed in the following sections. GUID generation requires a guarantee of uniqueness within the node ID for a given variant and version. Interoperability is provided by complying with the specified data structure. To prevent possible GUID collisions, which could be caused by different implementations on the same node, compliance with the algorithms specified here is required.

C.2.1   Clock Sequence

The clock sequence value must be changed whenever:
* The GUID generator detects that the local value of UTC has gone
  backward; this may be due to normal functioning of the DCE Time
  Service.
* The GUID generator has lost its state of the last value of UTC used,
  indicating that time \f2 may have gone backward; this is typically
  the case on reboot.

While a node is operational, the GUID service always saves the last UTC used to create a GUID. Each time a new GUID is created, the current UTC is compared to the saved value and if either the current value is less (the non-monotonic clock case) or the saved value was lost, then the clock sequence is incremented modulo 16,384, thus avoiding production of duplicate GUIDs.

The clock sequence must be initialized to a random number to minimize the correlation across systems. This provides maximum protection against node identifiers that may move or switch from system to system rapidly. The initial value MUST NOT be correlated to the node identifier.

The rule of initializing the clock sequence to a random value is waived if, and only if, all of the following are true:
* The clock sequence value is stored in a form of non-volatile storage.
* The system is manufactured such that the IEEE address ROM is designed to be inseparable from the system by either the user or field service, so that it cannot be moved to another system.
* The manufacturing process guarantees that only new IEEE address ROMs are used.
* Any field service, remanufacturing or rebuilding process that could change the value of the clock sequence must reinitialise it to a random value.

In other words, the system constraints prevent duplicates caused by possible migration of the IEEE address, while the operational system itself can protect against non-monotonic clocks, except in the case of field service intervention. At manufacturing time, such a system may initialise the clock sequence to any convenient value.

C.2.2   System Reboot

There are two possibilities when rebooting a system:
* The GUID generator states that the last UTC, adjustment, and clock sequence of the GUID service has been restored from non-volatile store.
* The state of the last UTC or adjustment has been lost.

If the state variables have been restored, the GUID generator just continues as normal. Alternatively, if the state variables cannot be restored, they are reinitialized, and the clock sequence is changed. If the clock sequence is stored in non-volatile store, it is incremented; otherwise, it is reinitialized to a new random value.

C.2.3   Clock Adjustment

GUIDs may be created at a rate greater than the system clock resolution. Therefore, the system must also maintain an adjustment value to be added to the lower-order bits of the time. Logically, each time the system clock ticks, the adjustment value is cleared. Every time a GUID is generated, the current adjustment value is read and

incremented atomically, and then added to the UTC time field of the
GUID.

C.2.4   Clock Overrun

The 100-nanosecond granularity of time should prove sufficient even for
bursts of GUID creation in the next generation of high-performance
multiprocessors. If a system overruns the clock adjustment by
requesting too many GUIDs within a single system clock tick, the GUID
service may raise an exception, handled in a system or process-
dependent manner either by:
* Terminating the requester.
* Reissuing the request until it succeeds.
* Stalling the GUID generator until the system clock catches up.

If the processors overrun the GUID generation frequently, additional
node identifiers and clocks may need to be added.

C.2.5   GUID Generation

GUIDs are generated according to the following algorithm:
* Determine the values for the UTC-based timestamp and clock sequence
  to be used in the GUID.
* Sections format and clock_seq define how to determine these values.
  For the purposes of this algorithm, consider the timestamp to be a
  60-bit unsigned integer and the clock sequence to be a 14-bit
  unsigned integer. Sequentially number the bits in a field, starting
  from 0 (zero) for the least significant bit.
* Set the time_low field equal to the least significant 32 bits (bits
  numbered 0 to 31 inclusive) of the time stamp in the same order of
  significance. If a DCE Security version GUID is being created, then
  replace the time_low field with the local user security attribute as
  defined by the *(ZB.
* Set the time_mid field equal to the bits numbered 32 to 47 inclusive
  of the timestamp in the same order of significance.
* Set the 12 least significant bits (bits numbered 0 to 11 inclusive)
  of the time_hi_and_version field equal to the bits numbered 48 to 59
  inclusive of the time stamp in the same order of significance.
* Set the 4 most significant bits (bits numbered 12 to 15 inclusive)
  of the time_hi_and_version field to the 4-bit version number
  corresponding to the GUID version being created, as shown in the
  table above.
* Set the clock_seq_low field to the 8 least significant bits (bits
  numbered 0 to 7 inclusive) of the clock sequence in the same order
  of significance.

* Set the 6 least significant bits (bits numbered 0 to 5 inclusive) of
  the clock_seq_hi_and_reserved field to the 6 most significant bits
  (bits numbered 8 to 13 inclusive) of the clock sequence in the same
  order of significance.
* Set the 2 most significant bits (bits numbered 6 and 7) of the
  clock_seq_hi_and_reserved to 0 and 1, respectively.
* Set the node field to the 48-bit IEEE address in the same order of
  significance as the address.

C.3   String Representation of GUIDs

For use in human-readable text, a GUID string representation is
specified as a sequence of fields, some of which are separated by
single dashes.

Each field is treated as an integer and has its value printed as a
zero-filled hexadecimal digit string with the most significant digit
first. The hexadecimal values a to f inclusive are output as lowercase
characters, and are case-insensitive on input. The sequence is the same
as the GUID constructed type.

The formal definition of the GUID string representation is provided by
the following extended BNF:

```
GUID                    = <time_low> <hyphen> <time_mid> <hyphen>
                          <time_high_and_version> <hyphen>
                          <clock_seq_and_reserved>
                          <clock_seq_low> <hyphen> <node>
time_low                = <hexOctet> <hexOctet> <hexOctet> <hexOctet>
time_mid                = <hexOctet> <hexOctet>
time_high_and_version   = <hexOctet> <hexOctet>
clock_seq_and_reserved  = <hexOctet>
clock_seq_low           = <hexOctet>
node                    = <hexOctet><hexOctet><hexOctet>
                          <hexOctet><hexOctet><hexOctet>
hexOctet                = <hexDigit> <hexDigit>
hexDigit                = <digit> | <a> | <b> | <c> | <d> | <e> | <f>
digit                   = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7"
|
                          "8" | "9"
hyphen                  = "-"
a                       = "a" | "A"
b                       = "b" | "B"
c                       = "c" | "C"
d                       = "d" | "D"
e                       = "e" | "E"
f                       = "f" | "F"
```

The following is an example of the string representation of a GUID:
2fac1234-31f8-11b4-a222-08002b34c003

C.4   Comparing GUIDs

The following table lists the GUID fields in order of significance,
from most significant to least significant, for purposes of GUID
comparison. The table also shows the data types applicable to the
fields.

```
+-------------------------------+------------------------------------+
| Field:                        | Type:                              |
+-------------------------------+------------------------------------+
| time_low                      | Unsigned 32-bit integer            |
| time_mid                      | Unsigned 16-bit integer            |
| time_hi_and_version           | Unsigned 16-bit integer            |
| clock_seq_hi_and_reserved     | Unsigned 8-bit integer             |
| clock_seq_low                 | Unsigned 8-bit integer             |
| node                          | Unsigned 48-bit integer            |
+-------------------------------+------------------------------------+
```

Consider each field to be an unsigned integer as shown above. Then, to
compare a pair of GUIDs, arithmetically compare the corresponding
fields from each GUID in order of significance and according to their
data type. Two GUIDs are equal if and only if all the corresponding
fields are equal. The first of two GUIDs follows the second if the most
significant field in which the GUIDs differ is greater for the first
GUID. The first of a pair of GUIDs precedes the second if the most
significant field in which the GUIDs differ is greater for the second
GUID.

C.5   Node IDs when no IEEE 802 network card is available

If a system wants to generate GUIDs but has no IEE 802-compliant
network card or other source of IEEE 802 addresses, then this section
describes how to generate one.

The ideal solution is to obtain a 47-bit cryptographic quality random
number, and use it as the low 47 bits of the node ID, with the high-
order bit of the node ID set to 1. (The high-order bit is the
unicast/multicast bit, which will never be set in IEEE 802 addresses
obtained from network cards.)

If a system does not have a primitive to generate cryptographic quality
random numbers, then in most systems there are usually a fairly large
number of sources of randomness available from which one can be
generated. Such sources are system-specific, but often include:
* the percent of memory in use

* the size of main memory in bytes
* the amount of free main memory in bytes
* the size of the paging or swap file in bytes
* free bytes of paging or swap file
* the total size of user virtual address space in bytes
* the total available user address space bytes
* the size of boot disk drive in bytes
* the free disk space on boot drive in bytes
* the current time
* the amount of time since the system booted
* the individual sizes of files in various system directories
* the creation, last read, and modification times of files in
  various system directories
* the utilization factors of various system resources (heap, and so
  on.)
* current mouse cursor position
* current caret position
* current number of running processes, threads
* handles or IDs of the desktop window and the active window
* the value of stack pointer of the caller
* the process and thread ID of caller
* various processor architecture specific performance counters
  (instructions executed, cache misses, TLB misses)

In addition, items such as the computer's name and the name of the
operating system, while not strictly speaking random, will
differentiate the results from those obtained by other systems.

The exact algorithm to generate a node ID using this data is system-
specific, because both the data available and the functions to
obtain them are often very system-specific. However, assuming that
one can concatenate all the values from the randomness sources into
a buffer, and that a cryptographic hash function such as MD5 [3] is
available, the following code will compute a node ID:

```
#include <md5.h>
#define HASHLEN 16

void GenNodeID(
        unsigned char * pDataBuf,       // concatenated "randomness values"
        long cData,                     // size of randomness values
        unsigned char NodeID[6]         // node ID
) {
        int i, j, k;
        unsigned char Hash[HASHLEN];
      MD_CTX context;
```

```
  MDInit (&context);
  MDUpdate (&context, pDataBuf, cData);
  MDFinal (Hash, &context);

        for (i,j = 0; i < HASHLEN; i++) {
                NodeID[j] ^= Hash[i];
                if (j == 6) j = 0;
        };
        NodeID[0] |= 0x80;                  // set the multicast bit
};
```

Other hash functions, such as SHA-1 [4], can also be used (in which case HASHLEN will be 20). The only requirement is that the result be suitably random - in the sense that the outputs from a set uniformly distributed inputs are themselves uniformly distributed, and that a single bit change in the input can be expected to cause half of the output bits to change.

C.6   Appendix C's References

[1] Lisa Zahn, et.al. Network Computing Architecture. Englewood
     Cliffs, NJ: Prentice Hall, 1990
[2] OSF DCE Spec
[3] R. Rivest, RFC 1321, "The MD5 Message-Digest Algorithm,"
     04/16/1992.
[4] SHA Spec