Revision 0.10

Editors David Pare, AOL
Art Howarth, Cisco Systems

Ultravox 3.0

A Media Agnostic Lightweight Streaming Protocol

Copyright 2004, America Online, Inc. and Cisco Systems, Inc.
All rights reserved.

Table of Contents

Abstract

The Ultravox 3 protocol provides for a simple and lightweight framing mechanism for interspersing status, codec data frames, and metadata frames for live as well as clip-based streaming. Connection setup closely resembles HTTP in form as well as response codes and headers. Optional configuration information is passed as arguments, while mandatory configuration data is passed in the headers. Like HTTP, Ultravox3 has been implemented using TCP. However, the codec data frames could be delivered via other transport protocols such as UDP.

In a live broadcasting Ultravox system, there are three components: a live Broadcaster that performs the encoding and potentially encrypts the stream, the Distribution Point that receives the encoded stream from the live broadcaster and acts as a high performance stream replicator, and a Listener that connects to the Distribution Point to obtain and play the stream.

In a clip-based Ultravox system, there are two components: a Distribution Point that wraps (and potentially encrypts) data into Ultravox Frames, and a Listener that connects to the Distribution Point to obtain and play the clip. Optionally, Distribution Points may choose not to wrap data in Ultravox Frames, and instead just send down the unmodified clip via http.

Ultravox 3 also provides for encrypting the metadata and data frames. Although Ultravox3 does not define any particular key exchange mechanism, it does have a suggested implementation.

What drove the implementation of Ultravox? Other streaming systems attempt to solve a wide variety of different problems. These implementations tend to be complicated, difficult to scale, and a challenge to implement properly. The simplicity of Ultravox3 and the tight focus it has on basic commercial streaming allows it to be much simpler to implement, and easier to optimize. There is no complicated negotiation setup, and when used in tandem with NSV (Nullsoft Video) format, the result is a simple and scalable architecture that has proven to be a very economical and well-performing distribution platform for the relatively straightforward task of delivering commercial audio and video streams over the Internet.

1 Ultravox Protocol

1.1 Overview

The Ultravox Protocol consists of two protocol sections: a Broadcaster section, and a Listener section. Overall, the protocol uses a request/response mechanism that is a subset of HTTP 1.1. Broadcasters use POST requests, listeners use GET requests, and the data is encapsulated in Ultravox frames. Both Broadcasters and Listeners connect to Ultravox Distribution Points using TCP.

An Ultravox Broadcaster connects to the Distribution Point and then sends an HTTP POST request. The Distribution Point responds using a 100-Continue. Then the Broadcaster sends the stream of data and broadcaster status Ultravox frames as the body of the HTTP post.

An Ultravox Listener connects to the Distribution Point, and then sends an HTTP GET request that describes the stream. The Distribution Point responds with a response header that describes server-specific values for the connection, and then sends Ultravox frames as the body of the response. Some streams continue in perpetuity, while other streams end. Listeners must be prepared to receive either type of stream. If a stream ends, the Ultravox Distribution Point will send a termination frame, and then close the TCP connection.

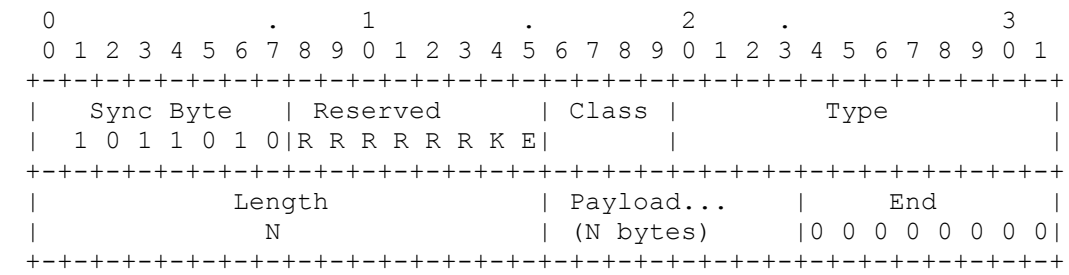
1.2 Ultravox Frames

The Body of Ultravox GET and POST messages consist of Ultravox Frames. An Ultravox frame, illustrated below, consists of a one-byte framing synchronization byte, a one-byte field reserved for future use, a four-byte message header, an optional payload containing a variable number of bytes, and a one-byte end of-message marker with the value of 0x00.

The four-byte Ultravox message header is divided into three fields that identify the message content and size. Note that the minimum message size is seven bytes – one synchronization byte, one reserved byte, four header bytes, and one end-of-message byte, while the maximum message size is 65542, allowing for a maximum payload length of 65535 bytes.

The three fields defined in the message header describe the message class, type (unique per class), and the length of the payload. The message class is described by four bits and it provides a simple categorization of messages to aid in their processing. The 12-bit message type identifies specific messages within a class. The 16-bit message payload length field allows for a maximum payload size of 65535 bytes. All fields are in network byte order (big-endian).

Ultravox Frame Header Format



Sync (8 bits) – Frame synchronization byte, always contains 0x5A.

Flags (8 bits) – first 7 bits reserved (marked **R)** must be zero. The low order **E** bit (bit 15) signifies that the message is encrypted, if the bit is set. Only metadata and codec data messages should be encrypted. The **K** bit (bit 14) signifies that the current frame contains a key or initial data codec frame.

Class (4 bits) – Classifies the use, nature, and purpose of the message:

0x1	Broadcaster to Distribution Point Messages	
0x2	Distribution Point to Listener Messages	
0x3-0x6	Metadata	Metadata messages
0x7-0xf	Pass-through Data	Data, typically Codec data

Type (12 bits) – class-specific use (e.g., which codec for data messages, which format for metadata messages, etc).

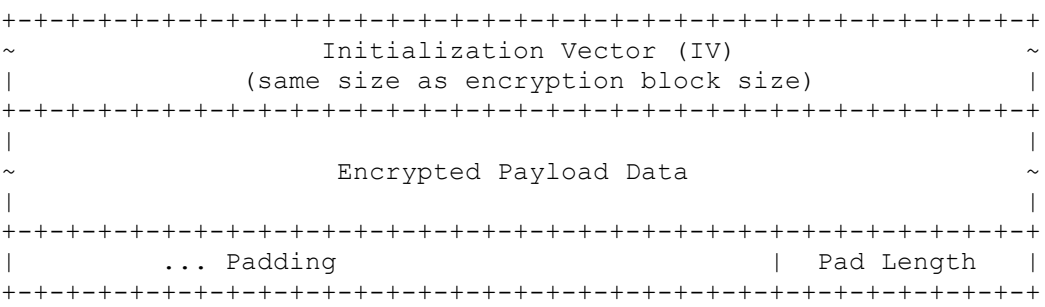
Length (16 bits) – number of bytes of payload (maximum of 65535); does not include trailing byte.

1.3 Encrypted Messages (Payload for any classes 3-15)

Ultravox Frames with the **E** bit (Flags field, bit 15) set are encrypted. Only the message payload is encrypted, and this is only suggested for metadata and codec data. Encrypted payloads will be larger than unencrypted payloads.

Ultravox assumes that Distribution Points may be able to pre-encrypt and cache content. To facilitate this, and to accommodate various trick play features such as fast fwd and seek, Ultravox assumes the use of a Cipher Block Chaining (CBC)-style encryption algorithm together with an Initialization Vector (IV). This method allows each individual frame to be decrypted on its own, while still providing a different encrypted block even if the data is identical.

CBC-style algorithms require extra space for Initialization Vectors. An IV is the same size as the encryption block size. In addition, since CBC algorithms require that all messages are a multiple of the block size there may be padding bytes required. As a result, an encrypted data payload comprises the following:



Some useful RFCs that discuss how CBC encryption works with variable length messages (such as IP datagrams) include RFC 1829 and RFC 3602.

- http://www.faqs.org/rfcs/rfc1829.html (use of 3DES in IPSEC)
- http://www.faqs.org/rfcs/rfc3602.html (use of AES in IPSEC).

Note that if a Distribution Point actually provides on-the-fly encryption, it must take into account of the maximum increase in payload size, and inform its broadcasters that it can only accept a payload of maxPayload – 2*blocksize.

Ultravox encryption assumes that keys can be uniquely identified by a KeyBlock GUID + index. KeyBlocks are identified by **GUID**, and may contain many different keys. The specific key in the KeyBlock is identified by the **index**. A new cacheable metadata message, the 0x3902 message, contains a small bit of XML that uniquely identifies the Key used to encrypt the stream, and will look as follows:

```
<KeySpecifier>
      <GUID>block-guid</GUID>
      <Index>number</Index>
</KeySpecifier>
```

If a given stream is encrypted, an Ultravox Distribution Point must start its stream with a 0x3902 metadata message (metadata messages are described later), since the follow-on metadata and the stream data itself may be encrypted. Encryption keys may be changed in mid stream. If a new 0x3902 metadata message arrives, it indicates that the frames that follow have been encrypted using the specified key.

Note that since metadata blocks may be encrypted, the 0x3902 message must be the first message sent by the broadcaster to the Distribution Point. Furthermore, the Distribution Point must replace the saved 0x3902 with any subsequent 0x3902 message it may receive.

2 Ultravox Broadcaster Protocol

2.1 POST Headers

To connect, Broadcaster must construct the following POST request, including an URL, a collection of required headers, a path, and some optional arguments.

```
POST
   <path>[?BufferSize=<size>&MaxPayload=<bytes>&ListenerAuth=<Y|N>]
   HTTP/1.1\r\n
Host: ultravox.aol.com\r\n
User-Agent: UvoxLive Ultravox Broadcaster 2.2\r\n
Expect: 100-continue\r\n
Ultravox-Protocol: 3.0\r\n
```

```
Ultravox-UID: <UID>\r\n
Ultravox-Auth-Token: <Authblob>\r\n
Ultravox-Auth-Profile: <AuthProfile>\r\n
Ultravox-Content-Type: <misc/ultravox>
Ultravox-Avg-Bitrate: <Avg Bit Rate>\r\n
Ultravox-Max-Bitrate: <Max Bit Rate>\r\n
Accept: */*\r\n
[more implementation-dependent headers]
\r\n
```

The <path> is an URL, and is implementation dependent. We only have one example:

- Radio @ AOL stream using a pseudo pathname: http://ultravox.aol.com/stream/<*SID*>, where SID is the Radio Station ID, typically a number such as 8001.

Some of these headers are required, and some are optional.

2.1.1 Required Headers

- **Expect: <100-continue>** - indicates the Distribution Point should respond with a 100-continue message instead of a 200 OK.
- **User-Agent: <Broadcaster Name Version X.Y>** - free form text the broadcaster supplies to identify itself. It must contain the word "ultravox" somewhere in the text.
- **Ultravox-Protocol: <3.0> –** ultravox protocol version number.
- **Ultravox-UID: <text> -** the username/ID of the broadcaster
- **Ultravox-Auth-Profile: <enum integer digits>**
- **Ultravox-Auth-Token: <depends on Auth-Profile> -** The UID, Auth-Token, and Auth-Profile are used in combination to perform authentication of the Broadcaster. AuthProfiles include 1:USA, 2:plaintext password. To obtain a USA AuthToken, a Broadcaster authenticates itself to an AppServer that knows how to generate AuthTokens. This AppServer constructs an AuthToken for the Listener using a secret that is shared between the AppServer and the Distribution Point. AuthTokens are URLEncoded. More information can be found about AuthTokens by reading the USA Specification.
- **Ultravox-Content-Type: <misc/ultravox> -** This specifies to the Distribution Point that the data will be a stream of Ultravox frames.
- **Ultravox-Avg-Bitrate: <number> -** This informs the Distribution Point what the average bitrate to expect from this broadcaster. **Avg-Bitrate** should not exceed **Max-Bitrate**. The **Avg-Bitrate** is used to allocate stream buffer sizes in the Distribution Point. (bits per second)
- **Ultravox-Max-Bitrate: <number> -** This informs the Distribution Point the maximum bitrate to expect from this broadcaster. (bits per second).

2.1.2 Optional Arguments

- **BufferSize=<seconds> -** This advises the Distribution Point to reserve **seconds * bitrate / 8** bytes for the listener ring buffer. The Distribution Point may decide to modify this value or ignore it entirely. It is advisory only.

- **MaxPayload= <bytes> -** This advises the Distribution Point that the Broadcaster wishes to send messages of at most <bytes> long. This is also advisory, since the Distribution Point is allowed to ultimately decide this value.
- **ListenerAuth=<Y/N> -** This instructs the Distribution Point to override its default settings for Listener Authentication: either require (Y) or disable (N) auth for this particular station.

2.2 Response Codes

If the Ultravox Distribution Point does not receive a complete HTTP GET request, or if the request does not contain a valid User-Agent header, the Listener is simply disconnected. Otherwise, the Ultravox Distribution Point responds to the request by sending the appropriate response as follows:

Summary of Ultravox Response Codes

Response	Description
HTTP/1.0 100 Continue	Stream is ready, Distribution Point is Responding with stream parameters.
HTTP/1.0 302 Redirect	Distribution Point suggests the Broadcaster connect to the supplied URL instead. The supplied URL is a base URL, and the Broadcaster must append arguments itself
HTTP/1.0 400 Bad Request\r\n\r\n	Could not parse the request, or bad range on an argument
HTTP/1.0 403 Forbidden\r\n\r\n	Authentication is on, and this Broadcaster was denied access.
HTTP/1.0 404 Not Found\r\n\r\n	Stream was not found in the Distribution Point's configuration
HTTP/1.0 503 Service Unavailable\r\n\r\n	Bandwidth limit reached on Distribution Point; Broadcaster should try again later.

2.3 Response Headers

Assuming the Ultravox Distribution Point responds with a 100 Continue accepting the connection, it responds to this request by sending the following HTTP response header:

```
HTTP/1.1 100 Continue\r\n
Server: Ultravox <protocol-version>\r\n
Ultravox-Buffer-Size: <seconds>\r\n
Ultravox-Max-Payload: <MaxPayload>\r\n
Ultravox-Max-Fragments: <number>\r\n
\r\n
```

Since both the **Buffer-Size** and **Max-Payload** arguments are advisory whose limits depend on the Distribution Point's internal resources and configuration, the Distribution responds with the actual values used for the stream.

- ***<protocol-version>*** is the Ultravox protocol version; currently at 3.0
- ***<Ultravox-Buffer-Size>*** is number of seconds of buffer space the Distribution Point allocated for this stream.
- <***Ultravox-Max-Payload***> is the maximum number of bytes the Distribution Point will accept in one Ultravox Frame from the Broadcaster.
- <***Max-Fragments***> is maximum number of metadata fragments that the Distribution Point will handle.

2.4 Ultravox Broadcaster Messages

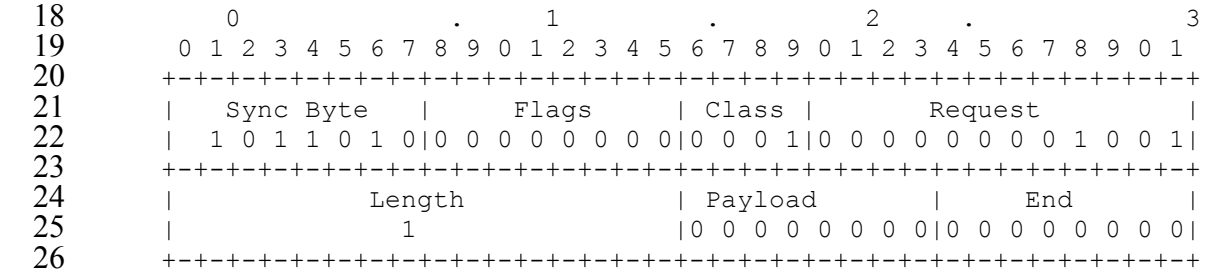
These are the valid Ultravox messages that can be sent from the Broadcaster to an Ultravox Distribution Point.

Important Note: The following Class 1 IDs are Reserved:

Class/Request: 0x1001, 0x1002, 0x1004, 0x1006, 0x1007, 0x1008

2.4.1 Temporary Broadcast Interruption

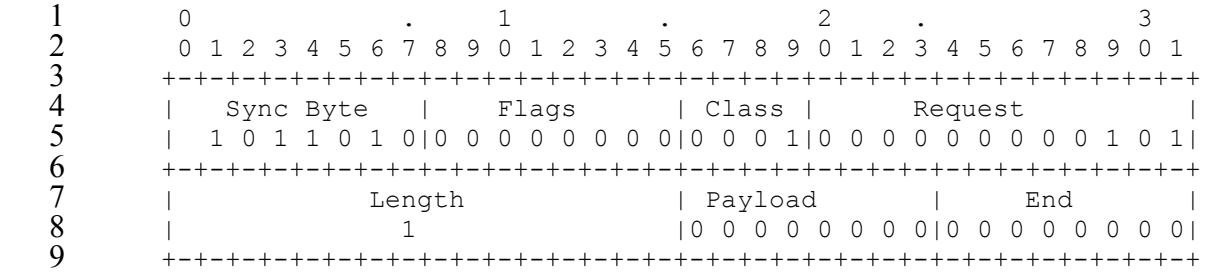
The Broadcaster uses this message to inform the Distribution Point that a temporary interruption has occurred in the stream to which it is distributing, and it is attempting to resolve this problem.



Class (4 bits) – 0x1
Request (12 bits) – 0x009
Length (16 bits) – 1.
Payload – 1 byte of null.

2.4.2 Broadcast Termination

When a Broadcaster sends this frame, it is indicating that no more messages will be sent, and the connection will be closed. The Ultravox Distribution Point will unsubscribe any listeners and terminate the stream, freeing its resources. This message has no payload. If a Broadcaster instead simply closes the TCP connection, the Ultravox Distribution Point will interpret this as an error, possibly logging messages and waiting for the Broadcaster to reconnect. The payload may only consist of zero or more bytes of NULL, up to a maximum length of 127. After sending the Broadcast Termination, the Ultravox Broadcaster will close the TCP connection. Properly written broadcasters will always send Broadcast Termination messages in the case of a normal termination of the broadcast.

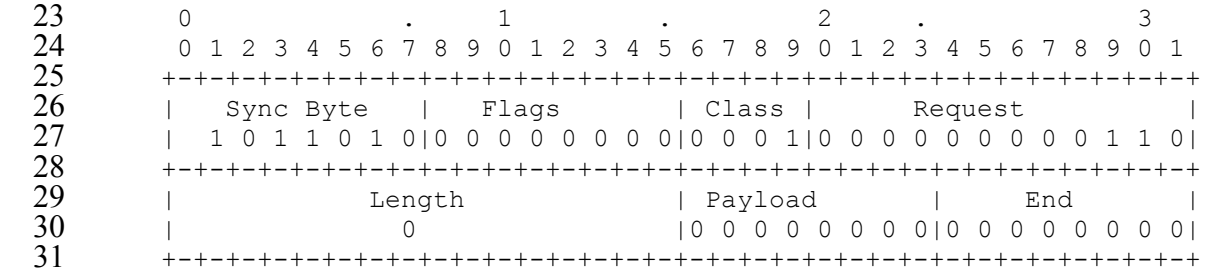


Class (4 bits) – 0x1
Notice (12 bits) – 0x005
Length (16 bits) – 1
Payload – one byte of null.

Upon receiving this message, the Ultravox Distribution Point first sends a Broadcast Termination message to all listeners that are currently subscribed to the stream. The Distribution point then closes the TCP connection to the Broadcaster and any remaining listener connections.

2.4.3 Flush Cached Metadata

When a Broadcaster sends this message, it causes the Ultravox Distribution Point to empty all cached metadata that it may be holding for the stream. This message has no payload.



Class (4 bits) – 0x1
Notice (12 bits) – 0x006
Length (16 bits) – 1
Payload – 1 byte of null.

Upon receiving this message, the Ultravox Distribution Point empties all cached metadata it is currently storing for the stream. If this message is received during a phase of the broadcast other than the data transfer phase, it is ignored by the Ultravox Distribution Point.

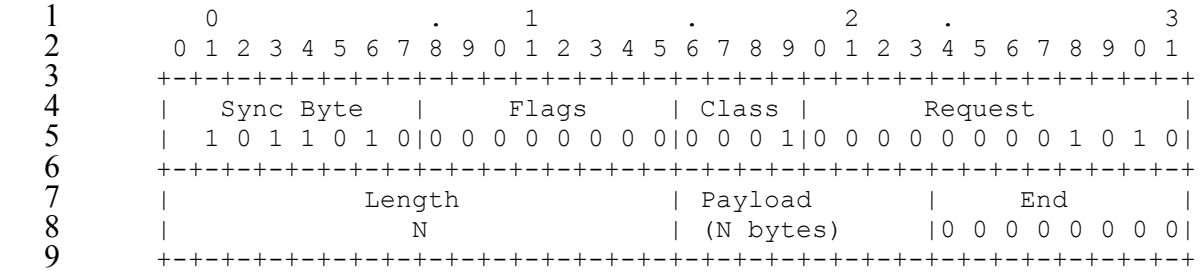
2.4.4 Broadcast Change

Lets the distribution point know who will take over broadcasting this stream w/o interrupting listener experience. Payload is identity of replacement broadcaster. This message is only used by Ultravox Broadcasters that have the ability to perform stream redirects. The Broadcaster uses this message to inform the Distribution Point that the current stream will be available from a different Broadcaster and that the current Broadcaster is going offline in a set period of time. After sending the remaining portion

of the stream, the Broadcaster closes the session with a Broadcast Termination message. The Distribution Point then switches to the second Broadcaster queue for transmission.

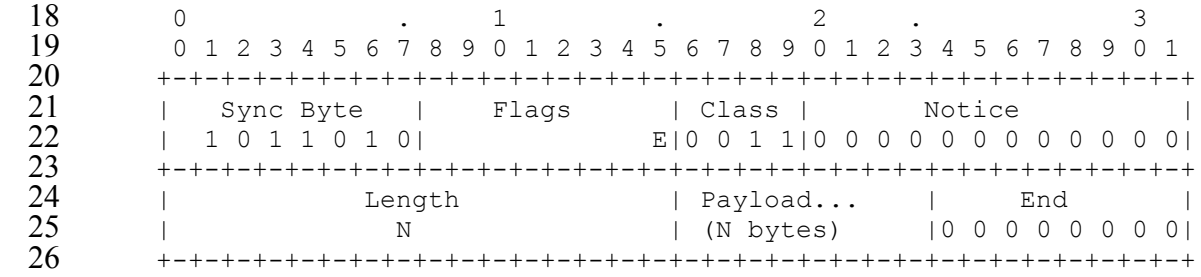
Note: No synchronization should be required by the Distribution Point, the remaining stream from the current Broadcaster should be immediately followed by the stream sent by the new broadcaster. A mechanism must exist so that the second Broadcaster fills the Distribution Point Buffer sufficiently to ensure that no gaps are introduced in playback and that the second Broadcaster knows when to start streaming to the Distribution Point.

Note: If the Distribution Point does not have the ability to accept stream redirects the Distribution Point simply disregards this packet and closes the session upon receipt of the Broadcast Termination message.



Class (4 bits) – 0x1
Request (12 bits) – 0x00a
Length (16 bits) – N
Payload:
Time Remaining (16 bits) – Time remaining in stream from current broadcaster (sec)
Path – Identity of new broadcaster will use when making connection to Distribution Point.

2.4.5 Metadata Messages (Class 3-6)



Class (4 bits) – 3-6
Notice (12 bits) –Metadata message type. Can be any number specific to the application.
Length (16 bits) – Number of bytes of payload
Metadata ID (16 bits) – a broadcaster-defined tag identifying the metadata (required, binary, network byte order).
Metadata Total Fragments (16 bits) – the number of fragments comprising this metadata ID (required, binary, network byte order). Note that ***MaxFragments*** sent back in the Response Header will indicate the maximum number of total fragments that a Distribution Point can handle.
Metadata Fragment Index (16 bits) – which metadata fragment message this is (required, binary, network byte order).
Metadata Payload – the metadata information (bytes, limited to ***MaxPayloadSize*** – 6).
A Broadcaster uses one or more messages of the same type within this class to compose a complete metadata package to supply the Listeners with information about the media currently being streamed.

There are two types of metadata messages – cacheable metadata, and pass-through metadata. Upon receiving a cacheable metadata message, a Distribution Point must maintain a cache of the most recently received messages based on their type, plus a cache of the current messages located in the stream buffer. When new listeners establish connections with this stream, all of the cached metadata messages are transmitted to the new listener as the first data sent to that listener.

In addition, after receiving any metadata message, the Distribution Point must pass it on to all existing listeners in band. Typically, the information contained is pertinent to changes in the content of the broadcast media (e.g., the current song now playing).

If a metadata message is larger than can fit in one Ultravox Frame, it is fragmented by the Broadcaster into several Ultravox Frames. Each message has an appropriate Fragment Index as well as a count of the total number of fragments in this metadata message. All of the metadata messages with a given ID are considered to comprise a single package of metadata.

Metadata messages should only be fragmented if they cannot fit within a single Ultravox frame.

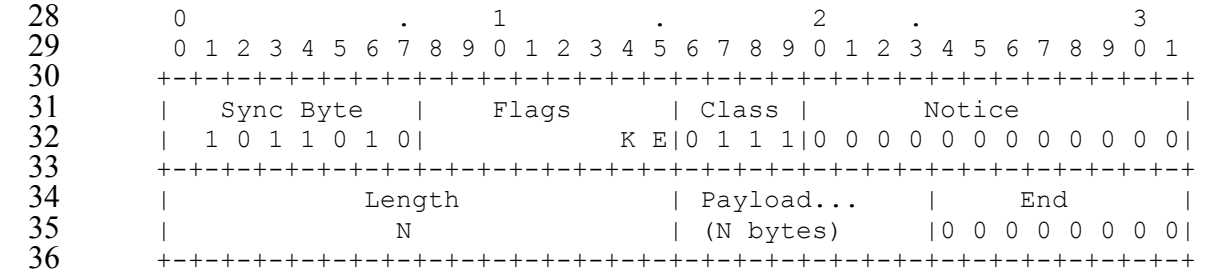
For a given Class/Notice value, if the Ultravox Distribution Point receives a message whose index is not already cached, it caches the message. However, if it receives a message whose Class/Notice value is already cached, the Ultravox Distribution Point must remove all the cached messages for that Class/Notice and then cache the new message.

The Notice field of a metadata message is used to identify type of metadata for the listener. For instance, the AOL Radio uses 0x3901 (type=0x3 cacheable metadata, notice=0x901) messages to provide song metadata in XML format, while it uses another message number to indicate how many seconds are left in a particular song currently playing. The specific values of metadata messages and their meanings and formats are service dependent.

The following metadata message classes should be followed:

0x3000	Content Information (Title…..)
0x3001	Information URL for current content
0x3901	Extended Content Information
0x5001	Time Remaining

2.4.6 Data Messages (Class 7-15)



Class (4 bits) – 0x7 – 0x15
Notice (12 bits) – 0x0-0xfff
Length (16 bits) – length of the data payload
Payload – data, typically codec or file format data

Messages of this class contain the actual media being broadcast on the stream. Most often these are used to pass codec data. The Broadcaster uses the Class/Type fields to identify specific codecs. For instance, a 0x7000 (class=0x7, notice=0x0) message is defined to be an MP3 data message. A list of well-known data types follows.

0x7000	MP3
0x7777	NSV
0x8000	Dolby VLB
0x8001	Ogg
0x8002	Speex

2.5 Broadcaster Message Flow

This section describes a sample sequence of messages by which a broadcaster establishes, maintains, utilizes, and terminates a media stream with the Ultravox Distribution Point.

This sequence is comprised of an exchange of messages between a Broadcaster and the Ultravox Distribution Point via a TCP connection that is established by the Broadcaster.

Once a TCP connection is established to an Ultravox Distribution Point, the Broadcaster sends an HTTP POST, the Ultravox Distribution Point responds with a 100 Continue, and then the Broadcaster transmits a stream of Ultravox data frames until terminating the broadcast with a 2005 Broadcast Terminated message.

Here is visual representation of a conversation between a Broadcaster and an Ultravox Distribution Point during a sample error-free session.

Broadcaster	Ultravox Distribution Point
→ POST /stream/1001	
	←100 Continue
→ 0x3901 CacheableMetadata:1:1:1:Song1	
→ 0x7000 DataMessage:<mp3 data>	
→ 0x7000 DataMessage:<mp3 data>	
→ 0x3901 CacheableMetadata:1:1:1:Song2	
→ 0x7000 DataMessage:<mp3 data>	
→ 0x2005 BroadcastTermination	
	Close TCP Connection to Broadcaster

3 Ultravox Listener Protocol

3.1 GET Headers

To obtain a stream, the Listener must construct a GET request, including an URL, and a User-Agent specifier.

```
GET <path>?{<tuple>{&<tuple>}} HTTP/1.0\r\n
Host: ultravox.aol.com\r\n
User-Agent: (User Agent Info)\r\n
Ultravox-protocol: 3.0\r\n
Ultravox-transport-type: TCP\r\n
```

```
Accept: */*\r\n
[more implementation-dependent headers]
\r\n
```

The <path> is an URL, and is implementation dependent. Some examples include:

- Radio @ AOL stream using a pseudo pathname: http://ultravox.aol.com/stream/<*SID*>, where SID is the Radio Station ID, typically a number such as 8001.
- Video @ AOL stream with a path terminating at /uvox, where the cachefile argument describes the name of the stream being streamed: http://nsvod.aol.com/uvox?cachefile=/aol/artists/universal/50cent/50cent_detroit_show_dvd_350.nsv

Typically, Application Servers are responsible for constructing an URL for a given piece of content. The Listener application provides a mechanism for the end user to select an URL, which is then passed on to the Distribution Point by the Listener to actually obtain the stream.

3.1.1 PrebufferTime

- PrebufferTime=<seconds>

Some distribution points will be sending streaming data output at the actual bitrate of the stream. For those distribution points, the PrebufferTime tuple specifies the amount of data (determined by the average bitrate multipled by the prebuffer time in seconds) that is sent *above* bitrate to the Listener immediately following the initial response. This enables the Listener to start the streaming experience as quickly as possible. In practice, we have observed that it is good to transmit the prebuffer at least four times the standard bitrate. If the Listener does not specify the prebuffer time, the Ultravox Distribution Point will use its default configured prebuffer time. In practice, 8 seconds has been a good default value.

3.1.2 Authentication

- **AuthToken**=<auth token>
- **AuthProfile**=<profileID>

Authentication is implemented using the AuthToken, and AuthProfile tuples. If the AuthToken is not specified, the Ultravox Distribution Point will substitute an empty string when attempting to authenticate the Listener. If AuthProfile is not specified, then USA authentication will be used to authenticate the Listener.

To obtain a USA AuthToken, a Listener typically authenticates itself to an AppServer that knows how to generate AuthTokens. This AppServer constructs an AuthToken for the Listener using a secret that is shared between the AppServer and the Distribution Point. AuthTokens are URLEncoded to allow them to be used as a value on a query string. More information can be found about AuthTokens by reading the USA Specification.

It is not required for a Distribution Point to implement authentication. If authentication is not implemented, any AuthToken, and AuthProfile arguments will be ignored.

3.1.3 Seeking and Truncating Playback

- **start**=<hh:mm:ss>
- **end**=<hh:mm:ss>

Seeking is implemented using **start**, while **end** is used for truncating playback before the end of the stream. These only make sense for Distribution Points that are able to seek back and forth within the content stream.

The start value informs the Distribution Point to start the streaming session at that offset. Note that for some content it is impossible to start the stream at exactly that time offset due to limitations of the encoding format. A well-written implementation will start streaming at the first complete data frame that is closest in time to the start time specified.

The end value tells the Distribution Point to stop streaming content whenever the stream reaches that time offset.

Both **start** and **end** are absolute time values relative to the start of the stream.

3.2 Response Codes

If the Ultravox Distribution Point does not receive a complete HTTP GET request, or if the request does not contain a valid User-Agent header, the Listener is simply disconnected. Otherwise, the Ultravox Distribution Point responds to the request by sending the appropriate response as follows:

Summary of Ultravox Response Codes

Response	Description
HTTP/1.0 200 Ok	Stream is ready, header and Data follows
HTTP/1.0 206 Partial Content	Seek request was processed properly, header and data follows
HTTP/1.0 302 Redirect	Distribution Point does not have the stream, but it suggests the Listener request it from the supplied URL. Listener must copy arguments.
HTTP/1.0 400 Bad Request\r\n\r\n	Could not parse the request, or bad range on an argument
HTTP/1.0 403 Forbidden\r\n\r\n	Authentication is on, and this Listener was denied access.
HTTP/1.0 404 Not Found\r\n\r\n	Stream was not found, or stream is not yet ready to stream.
HTTP/1.0 503 Service Unavailable\r\n\r\n	Bandwidth limit reached on Distribution Point; listener should try again later.

3.3 Response Headers

Assuming the Ultravox Distribution Point responds with a 200 OK (or a 206 Partial Content for a Range Request) accepting the connection, it responds to this request by sending the following HTTP response header:

```
HTTP/1.0 200 OK\r\n
Server: Ultravox <protocol-version>\r\n
Content-Type: <Content-Type>\r\n
Ultravox-Avg-Bitrate: <Avg Bit Rate>\r\n
Ultravox-Max-Bitrate: <Max Bit Rate>\r\n
Ultravox-Max-Fragments: <Max Fragment Count>\r\n
\r\n
```

The response headers have the following semantic meaning:

- ***<protocol-version>*** is the ultravox protocol version; currently at 3.0
- ***<Content-Type>*** is the content type, such as audio/mp3, audio/*, video/*, etc.
- <***AvgBit Rate***> is the target bitrate for the stream (bits per second).
- ***<Max Bit Rate***> is the maximum bitrate the stream will be sent at; for constant bitrate (CBR) stream, the average bitrate will be equal to the maximum bitrate. (bits per second).
- ***<Max Fragment Count***> is the maximum number of metadata fragments that a given metadata message will contain.
- ***<MaxMsgSize>*** is the maximum message size that the Distribution Point will send.

3.4 Ultravox Listener Messages

3.4.1 Temporary Broadcast Interruption

The Ultravox Distribution Point uses this message to inform the Listener that a temporary interruption has occurred in the stream to which it is subscribed, and it is attempting to resolve this problem by switching to a backup stream, starting a backup stream, and so on. This could be because the upstream live broadcaster has gone idle, or has disconnected without explicitly terminating the connection.

Listeners should ignore payload.

```
 0             .     1         .         2     .             3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Sync Byte   |    Flags      | Class |       Notice          |
|  1 0 1 1 0 1 0|0 0 0 0 0 0 0 0|0 0 1 0|0 0 0 0 0 0 0 0 0 0 0 1|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Length            | Payload...    |     End       |
|               N               | (N bytes)     |0 0 0 0 0 0 0 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

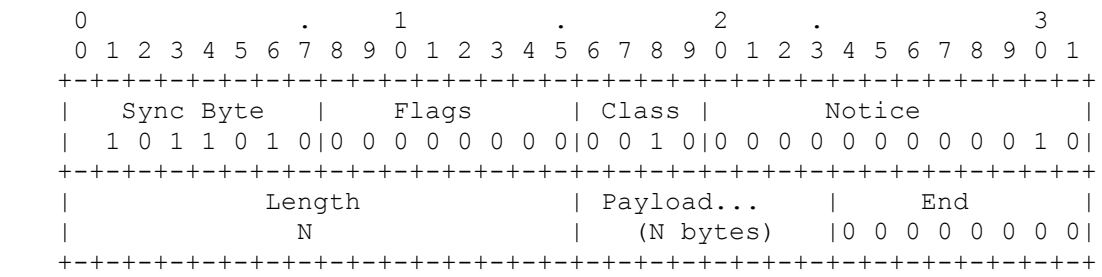
Class (4 bits) – 0x2
Notice (12 bits) – 0x001
Length (16 bits) – 0-127.
Payload – N bytes.

3.4.2 Broadcast Termination

The Ultravox Distribution Point uses this message to inform the Listener that the stream to which it is currently subscribed has terminated. This happens either when an upstream broadcaster has explicitly terminated the broadcast, or when the Ultravox Distribution Point has discovered a problem with an upstream source, and was unable to find a

fallback source within an implementation-defined period of time. After sending the Broadcast Termination, the Ultravox Distribution Point will close the TCP connection.

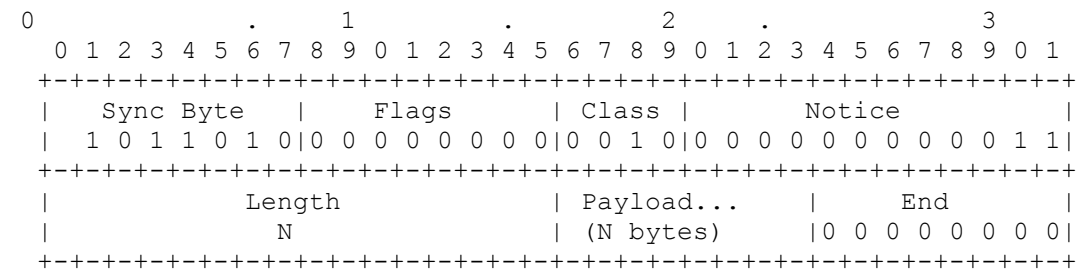
Listener should ignore payload.



Class (4 bits) – 0x2
Notice (12 bits) – 0x002
Length (16 bits) – N
Payload – N bytes

3.4.3 Broadcast Failover

The Ultravox Distribution Point uses this message to inform the Listener that the stream to which it had been subscribed has failed and a backup stream has been substituted for it. The payload contains a colon-separated string that describes the attributes of the new stream. This message is only used by Ultravox Distribution Points that have the ability to perform stream failovers.

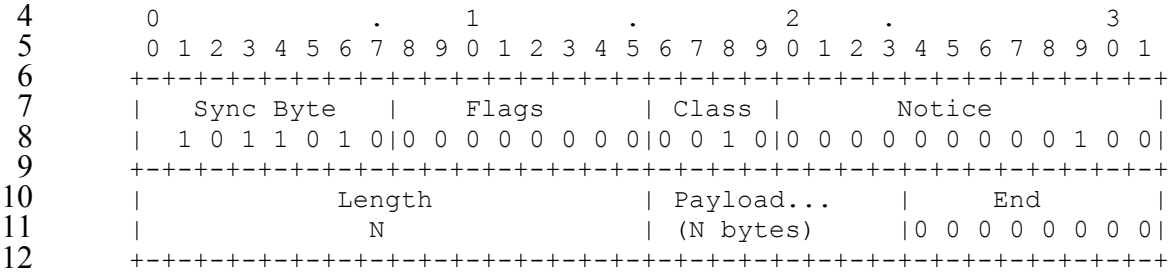


Class (4 bits) – 0x2
Notice (12 bits) – 0x003
Length (16 bits) – length of the payload string.
Payload – string containing the HTTP response header for the new stream, including `Server, Content-Type, Ultravox-Avg-Bitrate, Ultravox-Max-Bitrate, Ultravox-Max-Fragments.`
The payload values identify the characteristics of the backup stream that was substituted for the failed stream and to which the Listener is now subscribed.

3.4.4 Broadcast Discontinuity

The Ultravox Distribution Point uses this message to inform the Listener that the stream to which it had been subscribed has lost messages. Listeners may use this message to reset its codecs, stop presentation until it receives a new key frame, depending on particulars of the stream.

The Distribution Point sends this message when a listener underruns, or when it fails over to a stream with exactly the same stream definition (identical content, codec and bitrate) but from a different encoding source.



Class (4 bits) – 0x2
Notice (12 bits) – 0x004
Length (16 bits) – length of the payload string.
Payload – The listener may ignore the payload string.

3.5 Metadata and Data Messages

Distribution Points typically pass through Broadcaster Data and Metadata messages unchanged to the listener. Thus, the listener should be prepared to receive Data and Metadata messages as specified in the Broadcaster section of this document.

4.0 Document Modification History

Rev	Date	Originator	Comment
0.1	Jan 20, 2004	Art Howarth	Draft to define Outline
0.2	Mar 8, 2004	David Pare	First draft with Ultravox details
0.3	Mar 24, 2004	David Pare	Move to HTTP Post architecture + major modifications
0.4	Mar 30, 2004	Art Howarth	Add in Temporary Broadcaster Interruption, Modification to message lengths, + formatting
0.5	Mar 31, 2004	David Pare	Separate Required and Optional headers + minor verbage changes
0.6	April 9, 2004	Art Howarth	Add in Broadcaster Redirect, sync with existing AOL metadata message codes + formatting
0.7	April 26, 2004	David Pare	Added Encrypted Ultravox
0.8	May 5, 2004	David Pare	AOL Internal Review changes; re-merged Listener & broadcaster specifications
0.9	June 7, 2004	David Pare	Modified Broadcast Failover payload to contain HTTP response header
0.10	July 30, 2004	David Pare	Added Broadcast Discontinuity frame, and Key Frame header bit, acknowledgements.

5.0 Acknowledgements

Ultravox3 can trace its development back through three revisions, and then finally to the Shoutcast broadcasting protocol. We wish to acknowledge the contributions of those whose contributions helped make Ultravox3 a reality.

Protocol Development

- Shoutcast: Justin Frankel & Tom Pepper
- Ultravox1: Tom Pepper, Stephen Loomis, David Biderman
- Ultravox2: Stephen Loomis & David Biderman, Tom Pepper, Lance Richardson
- Ultravox3: David Pare, Art Howarth, Stephen Loomis

Product/Business/Legal/Operations:

- Ultravox1/2: Scott Brown, Chris Amen, Jim Bramson, John Schanz, Matt Korn, Joe Barrett, Gil Weigand, Bill Raduchel
- Ultravox3: Mike Wise, Scott Brown, David Bill, Jim Bramson, Alex Franco

6.0 Licenses

6.1 Copyright License

America Online, Inc. and Cisco Systems, Inc. grant you permission to reproduce and distribute this Specification, however, any copies you reproduce or distribute must be unmodified from the original, including this notice. All other rights are retained by the authors.

6.1 Patent License

America Online believes it may have certain patent rights that would be applicable to any fully compliant implementation of this Specification. In order to encourage the free and unrestricted development of implementations compliant with this Specification, America Online agrees to provide you with the following Patent License.

BY MAKING, USING, SELLING, OR OTHERWISE DISTRIBUTING A LICENSED IMPLEMENTATION, YOU AGREE TO THE TERMS OF THIS PATENT LICENSE.

1. Reciprocal Grants. America Online, Inc., on behalf of itself and its Affiliates, grants to you ("Licensee") a royalty-free, non-exclusive, non-transferable, non-sublicenseable, worldwide license under its Necessary Claims to make, use, sell, offer to sell, import, and otherwise distribute Licensed Implementations to other Licensed Entities and End Users, subject to a reciprocal grant by you, on behalf of yourself and your Affiliates, to America Online, Inc., its Affiliates, and all other Licensed Entities of a royalty-free, non-exclusive, non-transferable, non-sublicenseable, worldwide license under your Necessary Claims to make, use, sell, offer to sell, import, and otherwise distribute Licensed Implementations and to authorize such Licensed Implementations to be used by End Users.

2. No Other License. No license or right is granted except as expressly set forth herein.

3. Suspension/Termination. America Online, Inc. reserves the right to suspend or terminate this license if you or your Affiliate asserts that America Online, Inc., any of its Affiliates, or any other Licensed Entity infringes any patent claim as a result of or in connection with the use of this Specification or any Licensed Implementation. You have the right to suspend or terminate your reciprocal grant to any other Licensed Entity that asserts that you, any of your Affiliates, or any other Licensed Entity infringes any patent claim as a result of or in connection with the use of this Specification or any Licensed Implementation.

4. Definitions.

"Affiliate" means an entity that is directly or indirectly Controlled by a party to this agreement ("Party"), so long as such Control exists, where "Control" means beneficial ownership of more than fifty percent (50%) of the voting power or equity in an entity, or the direct or indirect right to manage the business affairs of an entity.

"End User" means any ultimate end user of a Licensed Implementation for its own personal use and not to develop, distribute, operate for the benefit of others, or use commercially.

"Licensed Entity" means any person or entity that has agreed to the terms of this Patent License.

"Licensed Implementation" means only those specific portions of an implementation that are fully compliant with this Specification.

"Necessary Claims" means those claims of all patents and patent applications, (other than design patents and design registrations), throughout the world, to which a Party or any Affiliate thereof has the right, at any time, to grant hereunder without such grant or the exercise of rights thereunder resulting in an obligation on the part of such Party or Affiliate to pay royalties or other consideration to any third party (except for payments to Affiliates or employees within the scope of their employment): (i) which are necessarily infringed by an implementation of this Specification where such infringement could not have been avoided by another commercially reasonable noninfringing implementation of this Specification, or (ii) for which infringement is based upon an implementation of any example included in the body of this Specification. Necessary Claims shall not include any claim other than as set forth above even if such claim is contained in the same patent as a Necessary Claim. Moreover, Necessary Claims shall not include claims that read on: (i) the implementation of separate published standards or specifications (including, for example and without limitation, MPEG4, IETF and other standards) which may be referred to, incorporated by reference, or included in this Specification, or (ii) enabling technologies (including, for example and without limitation, operating systems,

microprocessors, and software platforms) that may be necessary to make or use any product or portion thereof that complies with this Specification, but which are not expressly set forth in this Specification.

5. Disclaimer of Warranties; Limitation of Liability. THIS SPECIFICATION IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SPECIFICATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6. Miscellaneous. This License is made under, and shall be construed according to, the laws of the Commonwealth of Virginia, excluding its conflicts of laws principles. Any notice or other communication under this License shall be given in writing, if to America Online, Inc. at 22000 AOL Way, Dulles, VA 20166, and if to Licensee, at such address as America Online, Inc. can reasonably ascertain. In the event that any provision of this License conflicts with the law under which this License is to be construed or if any such provision is held invalid by a court with jurisdiction over the parties to this License, such provision shall be deemed to be restated to reflect the original intentions of the parties in accordance with applicable law, and the remainder of this License shall remain in full force and effect.