

ISMA
INTERNET STREAMING MEDIA ALLIANCE

Output Document Number **TD00105**

November 2007

TITLE: ISMA Ultravox Part 2: Ultravox 3.0 Protocol Specification
Status: Provisional Specification

Table of Contents

Abstract.....	3
1 Ultravox Protocol.....	4
1.1 Overview.....	4
1.2 Ultravox Frames.....	4
1.3 Encrypted Messages (Payload for any classes 3-15).....	5
2 Ultravox Broadcaster Protocol.....	6
2.1 POST Headers.....	6
2.1.1 Required Headers.....	7
2.1.2 Optional Arguments.....	7
2.2 Response Codes.....	7
2.3 Response Headers.....	8
2.4 Ultravox Broadcaster Messages.....	8
2.4.1 Temporary Broadcast Interruption.....	9
2.4.2 Broadcast Termination.....	9
2.4.3 Flush Cached Metadata.....	10
2.4.4 Broadcast Change.....	10
2.4.5 Metadata Messages (Class 3-6).....	11
2.4.6 Data Messages (Class 7-15).....	12
2.5 Broadcaster Message Flow.....	13
3 Ultravox Listener Protocol.....	13
3.1 GET Headers.....	13
3.1.1 PrebufferTime.....	14
3.1.2 Authentication.....	14
3.1.3 Seeking and Truncating Playback.....	15
3.2 Response Codes.....	15
3.3 Response Headers.....	15
3.4 Ultravox Listener Messages.....	16
3.4.1 Temporary Broadcast Interruption.....	16
3.4.2 Broadcast Termination.....	16
3.4.3 Broadcast Failover.....	17
3.4.4 Broadcast Discontinuity.....	17
3.5 Metadata and Data Messages.....	18
4 Document Modification History.....	19
5 Acknowledgements.....	20
6 Licenses.....	20
6.1 Copyright License.....	20
6.2 Patent License.....	20

1 **Abstract**

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

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

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

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

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

31

1 Ultravox Protocol

2 1.1 Overview

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

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

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

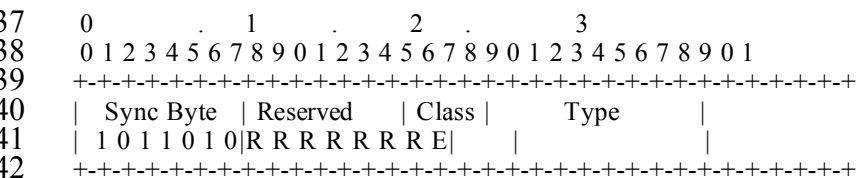
19 1.2 Ultravox Frames

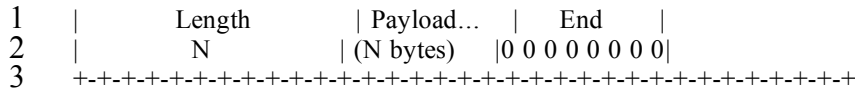
20 The Body of Ultravox GET and POST messages consist of Ultravox Frames. An
21 Ultravox frame, illustrated below, consists of a one-byte framing synchronization byte, a
22 one-byte field reserved for future use, a four-byte message header, an optional payload
23 containing a variable number of bytes,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

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

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

36 Ultravox Frame Header Format





- 4 **Sync** (8 bits) – Frame synchronization byte, always contains 0x5A.
- 5 **Flags** (8 bits) – first 7 bits reserved (marked **R**) must be zero. The low order **E** bit (bit
- 6 15) signifies that the message is encrypted, if the bit is set. Only metadata and codec
- 7 data messages should be encrypted.
- 8 **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

- 10
- 11 **Type** (12 bits) – class-specific use (e.g., which codec for data messages, which format for
- 12 metadata messages, etc).
- 13 **Length** (16 bits) – number of bytes of payload (maximum of 65535); does not include
- 14 trailing byte.

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

16 Ultravox Frames with the **E** bit (Flags field, bit 15) set are encrypted. Only the message

17 payload is encrypted, and this is only suggested for metadata and codec data. Encrypted

18 payloads will be larger than unencrypted payloads.

19 Ultravox assumes that Distribution Points may be able to pre-encrypt and cache content.

20 To facilitate this, and to accommodate various trick play features such as fast fwd and

21 seek, Ultravox assumes the use of a Cipher Block Chaining (CBC)-style encryption

22 algorithm together with an Initialization Vector (IV). This method allows each individual

23 frame to be decrypted on its own, while still providing a different encrypted block even if

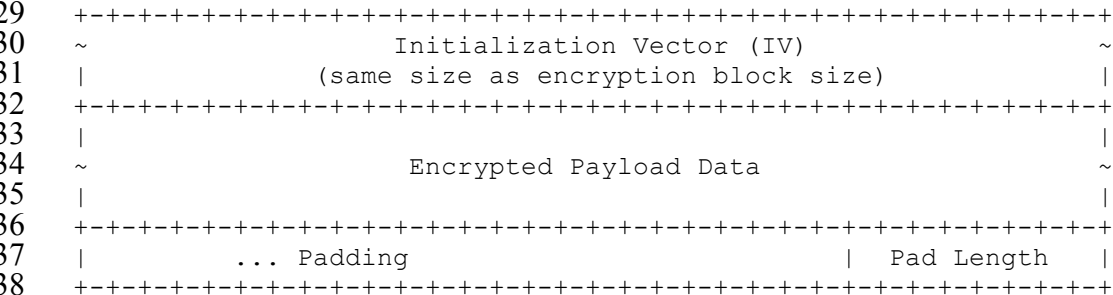
24 the data is identical.

25 CBC-style algorithms require extra space for Initialization Vectors. An IV is the same

26 size as the encryption block size. In addition, since CBC algorithms require that all

27 messages are a multiple of the block size there may be padding bytes required. As a

28 result, an encrypted data payload comprises the following:



39 Some useful RFCs that discuss how CBC encryption works with variable length

40 messages (such as IP datagrams) include RFC 1829 and RFC 3602.

- 41 • <http://www.faqs.org/rfcs/rfc1829.html> (use of 3DES in IPSEC)

1 • <http://www.faqs.org/rfcs/rfc3602.html> (use of AES in IPSEC).
2 Note that if a Distribution Point actually provides on-the-fly encryption, it must take into
3 account of the maximum increase in payload size, and inform its broadcasters that it can
4 only accept a payload of $\text{maxPayload} - 2 * \text{blocksize}$.

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

```
10 <KeySpecifier>  
11     <GUID>block-guid</GUID>  
12     <Index>number</Index>  
13 </KeySpecifier>
```

14 If a given stream is encrypted, an Ultravox Distribution Point must start its stream with a
15 0x3902 metadata message (metadata messages are described later), since the follow-on
16 metadata and the stream data itself may be encrypted. Encryption keys may be changed
17 in mid stream. If a new 0x3902 metadata message arrives, it indicates that the frames that
18 follow have been encrypted using the specified key.

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

23 **2 Ultravox Broadcaster Protocol**

24 **2.1 POST Headers**

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

```
27 POST  
28     <path>[?BufferSize=<secs>&MaxPayload=<bytes>&ListenerAuth=<Y|N>]  
29 HTTP/1.1\r\n  
30 Host: ultravox.aol.com\r\n  
31 User-Agent: UvoxLive Broadcaster 2.2\r\n  
32 Expect: 100-continue\r\n  
33 Ultravox-Protocol: 3.0\r\n  
34 Ultravox-UID: <UID>\r\n  
35 Ultravox-Auth-Token: <Authblob>\r\n  
36 Ultravox-Auth-Profile: <AuthProfile>\r\n  
37 Ultravox-Content-Type: <misc/ultravox>  
38 Ultravox-Avg-Bitrate: <Avg Bit Rate>\r\n  
39 Ultravox-Max-Bitrate: <Max Bit Rate>\r\n  
40 Accept: */*\r\n  
41 [more implementation-dependent headers]  
42 \r\n
```

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

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

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

2 **2.1.1 Required Headers**

- 3 • **Expect:** <100-continue> - indicates the Distribution Point should respond with a
4 100-continue message instead of a 200 OK.
- 5 • **User-Agent:** <Broadcaster Name Version X.Y> - free form text the broadcaster
6 supplies to identify itself
- 7 • **Ultravox-Protocol:** <3.0> – ultravox protocol version number.
- 8 • **Ultravox-UID:** <text> - the username/ID of the broadcaster
- 9 • **Ultravox-Auth-Profile:** <enum integer digits>
- 10 • **Ultravox-Auth-Token:** <depends on Auth-Profile> - The UID, Auth-Token,
11 and Auth-Profile are used in combination to perform authentication of the
12 Broadcaster. AuthProfiles include 1:USA, 2:plaintext password. To obtain a
13 USA AuthToken, a Broadcaster authenticates itself to an AppServer that knows
14 how to generate AuthTokens. This AppServer constructs an AuthToken for the
15 Listener using a secret that is shared between the AppServer and the Distribution
16 Point. AuthTokens are URLEncoded. More information can be found about
17 AuthTokens by reading the USA Specification.
- 18 • **Ultravox-Content-Type:** <misc/ultravox> - This specifies to the Distribution
19 Point that the data will be a stream of Ultravox frames.
- 20 • **Ultravox-Avg-Bitrate:** <number> - This informs the Distribution Point what the
21 average bitrate to expect from this broadcaster. **Avg-Bitrate** should not exceed
22 **Max-Bitrate**. The **Avg-Bitrate** is used to allocate stream buffer sizes in the
23 Distribution Point. (bits per second)
- 24 • **Ultravox-Max-Bitrate:** <number> - This informs the Distribution Point the
25 maximum bitrate to expect from this broadcaster. (bits per second).

26 **2.1.2 Optional Arguments**

- 27 • **BufferSize=<seconds>** - This advises the Distribution Point to reserve **seconds *
28 bitrate / 8** bytes for the listener ring buffer. The Distribution Point may decide to
29 modify this value or ignore it entirely. It is advisory only.
- 30 • **MaxPayload=<bytes>** - This advises the Distribution Point that the Broadcaster
31 wishes to send messages of at most <bytes> long. This is also advisory, since the
32 Distribution Point is allowed to ultimately decide this value.
- 33 • **ListenerAuth=<Y/N>** - This instructs the Distribution Point to override its
34 default settings for Listener Authentication: either require (Y) or disable (N) auth
35 for this particular station.

36 **2.2 Response Codes**

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

41 **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.

1 **2.3 Response Headers**

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

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

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

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

20

21 **2.4 Ultravox Broadcaster Messages**

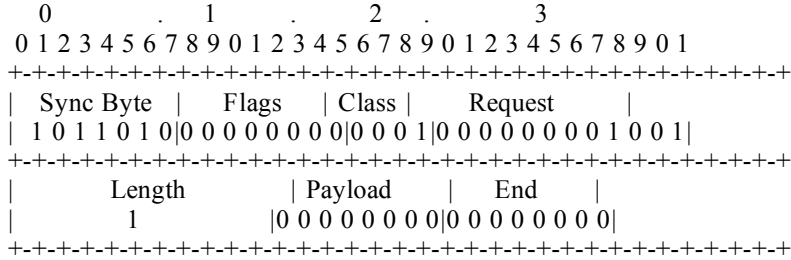
22 These are the valid Ultravox messages that can be sent from the Broadcaster to an
23 Ultravox Distribution Point.

24 Important Note: The following Class 1 IDs are Reserved:

25 Class/Request: 0x1001, 0x1002, 0x1003, 0x1004, 0x1007, 0x1008

2.4.1 Temporary Broadcast Interruption

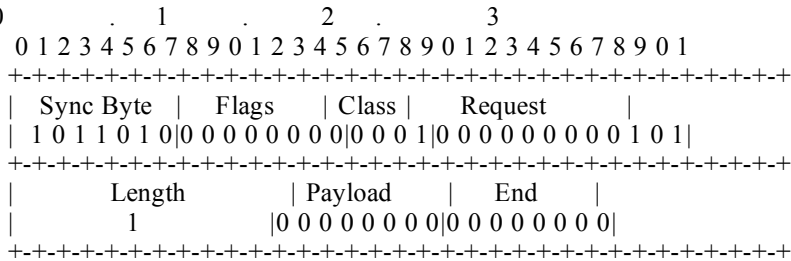
The Broadcaster uses this message to inform the Distribution Point that a temporary interruption has occurred in the stream to which it is distributing, and it is attempting to resolve this problem.



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

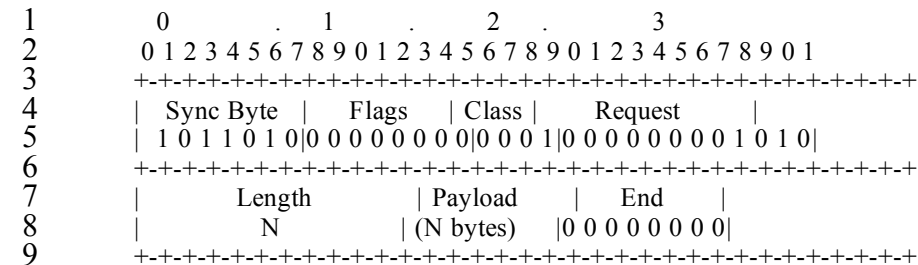
2.4.2 Broadcast Termination

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.



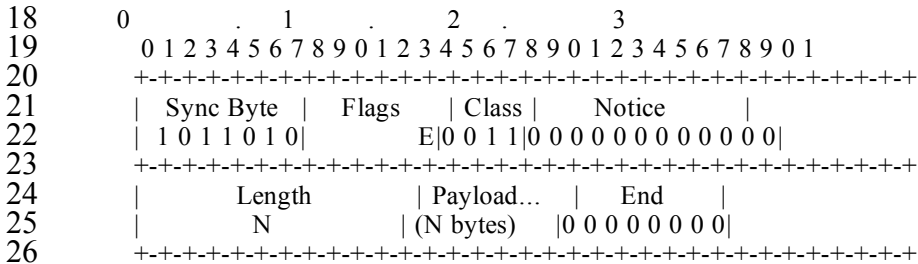
- Class** (4 bits) – 0x1
- Notice** (12 bits) – 0x005
- Length** (16 bits) – 1
- Payload** – one byte of null.

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.



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

17 **2.4.5 Metadata Messages (Class 3-6)**



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

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

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

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

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

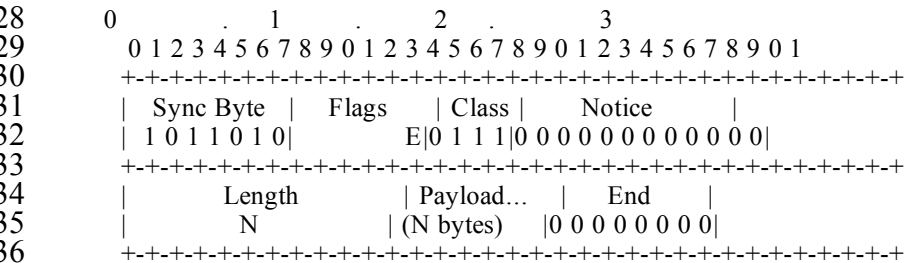
11 For a given Class/Notice value, if the Ultravox Distribution Point receives a message
 12 whose index is not already cached, it caches the message. However, if it receives a
 13 message whose Class/Notice value is already cached, the Ultravox Distribution Point
 14 must remove all the cached messages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

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

22 The following metadata message classes should be followed:

- 23 0x3000 Content Information (Title.....)
- 24 0x3001 Information URL for current content
- 25 0x3901 Extended Content Information
- 26 0x5001 Time Remaining

27 **2.4.6 Data Messages (Class 7-15)**



37 **Class** (4 bits) – 0x7 – 0x15

38 **Notice** (12 bits) – 0x0-0xffff

39 **Length** (16 bits) – length of the data payload

40 **Payload** – data, typically codec or file format data

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

- 45 0x7000 MP3

1 0x7777 NSV
 2 0x8000 Dolby VLB
 3 0x8001 Ogg
 4 0x8002 Speex
 5

6 **2.5 Broadcaster Message Flow**

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

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

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

15 Here is visual representation of a conversation between a Broadcaster and an Ultravox
 16 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

17

18 **3 Ultravox Listener Protocol**

19 **3.1 GET Headers**

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

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

1 [more implementation-dependent headers]
2 \r\n

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

- 4 • Radio @ AOL stream using a pseudo pathname:
5 <http://ultravox.aol.com/stream/<SID>>, where SID is the Radio Station ID,
6 typically a number such as 8001.
- 7 • Video @ AOL stream with a path terminating at /uvox, where the cachefile
8 argument describes the name of the stream being streamed:
9 [http://nsvod.aol.com/uvox?cachefile=/aol/artists/universal/50cent/50cent_detroit_](http://nsvod.aol.com/uvox?cachefile=/aol/artists/universal/50cent/50cent_detroit_show_dvd_350.nsv)
10 [show_dvd_350.nsv](http://nsvod.aol.com/uvox?cachefile=/aol/artists/universal/50cent/50cent_detroit_show_dvd_350.nsv)

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

15 3.1.1 PrebufferTime

- 16 • **PrebufferTime**=<seconds>

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

26 3.1.2 Authentication

- 27 • **AuthToken**=<auth token>
- 28 • **AuthProfile**=<profileID>

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

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

39 It is not required for a Distribution Point to implement authentication. If authentication is
40 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:

```

1 HTTP/1.0 200 OK\r\n
2 Server: Ultravox <protocol-version>\r\n
3 Content-Type: <Content-Type>\r\n
4 Ultravox-Avg-Bitrate: <Avg Bit Rate>\r\n
5 Ultravox-Max-Bitrate: <Max Bit Rate>\r\n
6 Ultravox-Max-Fragments: <Max Fragment Count>\r\n
7 Ultravox-Max-Msg: <Max Msg Size>\r\n
8 \r\n

```

9 The response headers have the following semantic meaning:

- 10 • <protocol-version> is the ultravox protocol version; currently at 3.0
- 11 • <Content-Type> is the content type, such as audio/mp3, audio/*, video/*, etc.
- 12 • <Avg Bit Rate> is the target bitrate for the stream (bits per second).
- 13 • <Max Bit Rate> is the maximum bitrate the stream will be sent at; for constant
- 14 bitrate (CBR) stream, the average bitrate will be equal to the maximum bitrate.
- 15 (bits per second).
- 16 • <Max Fragment Count> is the maximum number of metadata fragments that a
- 17 given metadata message will contain.
- 18 • <Max Msg Size> is the maximum message size that the Distribution Point will
- 19 send.

20 3.4 Ultravox Listener Messages

21 3.4.1 Temporary Broadcast Interruption

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

27 Listeners should ignore payload.

```

28 0          . 1          . 2          . 3
29 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
30 +-+-+-+-+-+-+-+-+
31 | Sync Byte | Flags | Class | Notice |
32 | 1 0 1 1 0 1 0|0 0 0 0 0 0 0|0 0 1 0|0 0 0 0 0 0 0 0 0 0 1|
33 +-+-+-+-+-+-+-+-+
34 | Length | Payload... | End |
35 | N | (N bytes) | 0 0 0 0 0 0 0|
36 +-+-+-+-+-+-+-+-+

```

37 **Class** (4 bits) – 0x2

38 **Notice** (12 bits) – 0x001

39 **Length** (16 bits) – 0-127.

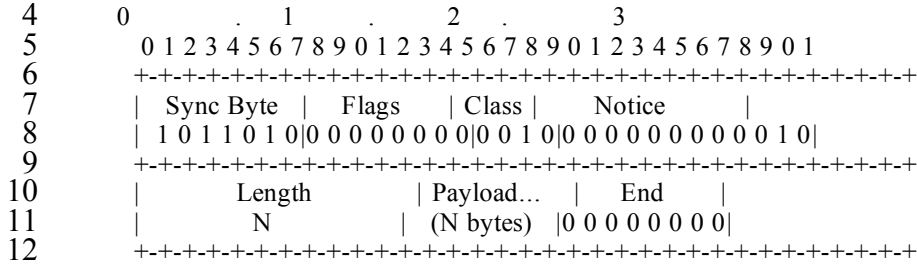
40 **Payload** – N bytes.

41 3.4.2 Broadcast Termination

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

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

3 Listener should ignore payload.



13 **Class** (4 bits) – 0x2

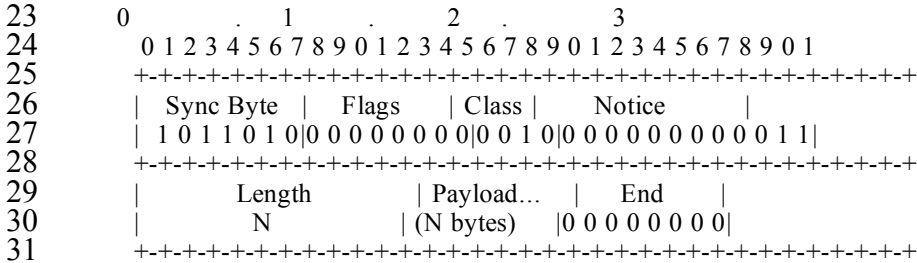
14 **Notice** (12 bits) – 0x002

15 **Length** (16 bits) – N

16 **Payload** – N bytes

17 **3.4.3 Broadcast Failover**

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



32 **Class** (4 bits) – 0x2

33 **Notice** (12 bits) – 0x003

34 **Length** (16 bits) – length of the payload string.

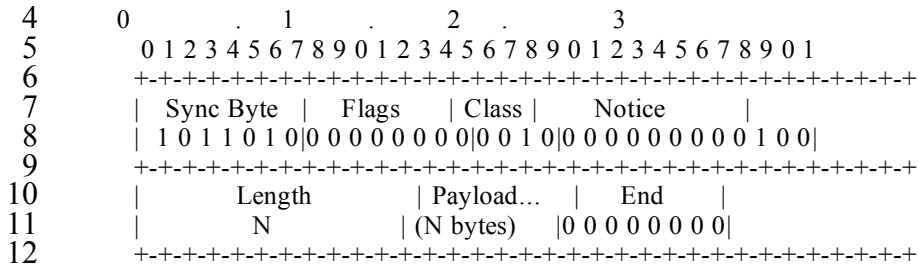
35 **Payload** – string containing the HTTP response header for the new stream, including
 36 Server, Content-Type, Ultravox-Avg-Bitrate, Ultravox-Max-Bitrate,
 37 Ultravox-Max-Fragments.

38 The payload values identify the characteristics of the backup stream that was substituted
 39 for the failed stream and to which the Listener is now subscribed.

41 **3.4.4 Broadcast Discontinuity**

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

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



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

17 **3.5 Metadata and Data Messages**

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

4 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.
0.11	November 28, 2005	Abiyu Diro	Fixed a handful of typos and omissions (include Ultravox-Max-Msg header)

5 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 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.

6.2 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 NONINFRINGEMENT 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.