SHOUTcast 2 (Ultravox 2.1) Protocol Details

From Winamp Developer Wiki

Shoutcast Home | Shoutcast Server (DNAS) | Shoutcast Developer (API) | Shoutcast For Business & Revenue Generation | Shoutcast DSP (encoder Plug-In for Winamp)

Contents

Overview

The SHOUTcast 2 / Ultravox 2.1 Protocol is an application level streaming protocol that will encapsulate stream data in order to abstract the underlying data encoding. The protocol defines the structure of the encapsulation as well as the series of handshakes necessary for setting up a streaming session. It describes the handshaking necessary between the broadcaster and the distribution point as well as the handshaking necessary between the listener and the distribution point. At this time the protocol is not codec dependent, nor is it transport dependent. The protocol is designed with network performance, compatibility, and simplicity in mind.

At a high level the broadcaster opens a connection to the distribution point. It provides authentication information and stream details to the distribution point. The distribution point parses the provided information, authenticates the user and then either sets up for streaming the broadcast feed, or denies the broadcaster access. Similarly, a listener may connect to the distribution point, provide authentication information, and request a particular stream. The distribution point will either allow or deny the listener and if allowed will start sending stream data to the listener from it’s internal stream buffer.

Recommendations for features of the distribution point buffering scheme and connection handling are also mentioned here, but are an implementation detail and the protocol may lend itself for better solutions based on the streaming architecture.

Ultravox Messages

The mechanism by which Ultravox is able to abstract the underlying stream encoding is to use encapsulation within a simple message structure. Once a broadcaster or listener is connected to the distribution point, Ultravox messages will be the sole communication mechanism.

The basic Ultravox message format is:

1	2	3	4	5	5	6	7
0101 1010	AAAA BBBB	CCCC DDDD	DDDD DDDD	EEEE EEEE	EEEE EEEE	F……...F	0000 0000
Sync	Res QoS	Class Type		Payload Length	Payload Length	Payload	0x00
[Sync byte 0x5A]	[Reserved/QoS]	[4 bit msg class]	[12 bit msg type]	[16 bit msg length - first half]	[16 bit msg length - second half]	[N byte payload]	[0x00 byte]

Note: All values of the message are network byte order.

Sync Byte (0x5A)

The sync byte is used to distinguish the beginning of an Ultravox message. It allows the listener to seek to a potential valid frame if data gets corrupted between the distribution point and the listener.

Reserved (A) and QoS (B)

QoS information may be passed in the low 4 bits of the reserve byte. The QoS is a four bit value whose high bit specifies whether this packet has to be delivered (useful for UDP). The other 3 bits are the relative priority for send queue. The high 4 bits are currently unused.

Message Class (C)

The message class is used in combination with the message type to uniquely identify an Ultravox message. The reason for separating the unique identifier into two units is for performance reasons. The message classes are defined so that the distribution point can look at the message class and decide whether it needs to examine the payload. Examining the payload requires extra parsing on the part of the distribution point and so will affect the performance when we talk about heavy load. The message classes are defined as:

Class	Type	Description
0x0	Operations	Not Defined
0x1	Broadcaster	Broadcaster Distribution Point
0x2	Listener	Distribution Point Listener
0x3	Cacheable Metadata	Broadcaster Listener via Distribution Point
0x4	Cacheable Metadata	Broadcaster Listener via Distribution Point
0x5	Pass-through Metadata	Broadcaster Listener via Distribution Point
0x6	Pass-through Metadata	Broadcaster Listener via Distribution Point
0x7	Data	Encoded data (ex: 0x7000 is MP3)
0x8	Data	Encoded data (ex: 0x8003 is AACP)
0x9	Framed Data	Encoded data (ex: 0x9000 is Headerless AACP)
0xA	Cacheable Binary Metadata	(Reserved for use with 0x9 if implemented)
0xB-0xF	N/D	Not Defined

See 'Known Message Types' for a complete list of supported or reserved message classes as well as the types related to them.

Message Type (D)

The message type is 12 bits in the message header that specify the type of data encapsulated in the message. See 'Broadcast Messages Table' for details about the specific message types supported.

Message Length (E)

The message length is the fifth and the sixth byte of the message header. It specifies the length of the payload following the message header.

```
The message length does not include the trailer 0x00 byte.
```

Payload (F)

The payload contains the data for each message. The payload size is not restricted by the protocol. However, the protocol specifies how a broadcaster, distribution point, and listener need to negotiate and report the maximum size message that will be sent in a single datastream. Some implementations may require that the messages stay under MTU, in which case the distribution point could enforce this during max-payload-size negotiation.

Trailing 0x00

The trailing 0x00 in the payload is used by the listener to determine if a message is malformed.

Broadcast Messages

The broadcast protocol describes the message requests from the broadcaster to the distribution point and responses by the distribution point. The messages are used to authenticate the broadcaster, deliver details about the stream to be broadcasted, manage the stream, and terminate the stream.

For most requests made by a broadcaster, the distribution point normally sends the same message class and type back to the broadcaster as a response, indicating success or failure of the request in the encapsulated payload.

The message types shown require a payload formatted according to the following criteria:

- The existence of payload data is indicated by a message length > 0
- All payloads for broadcaster messages are ASCIIZ strings (i.e. ASCII-encoded and terminated by a '\0').
- Parameters within the payload are separated by a single colon (":") character

Broadcast Messages Table

Broadcast messages are defined as a 12 bit request desired to be performed:

Class	Type	Description	Required?	Payload?	Response?
0x1	0x000	N/D	N/A	N/A	N/A
0x1	0x001	Authenticate Broadcast	Yes	Yes	Yes
0x1	0x002	Setup Broadcast	Yes	Yes	Yes
0x1	0x003	Negotiate Buffer Size	Yes	Yes	Yes
0x1	0x004	Standby	Yes	No	Yes
0x1	0x005	Terminate	No	No	No
0x1	0x006	Flush Cached Metadata	No	No	No
0x1	0x007	Require Listener Auth	No	Yes	Yes
0x1	0x008	Negotiate Max Payload Size	Yes	Yes	Yes
0x1	0x009	Request Cipher	Yes	No	Yes
0x1	0x040	Stream Mime Type	Yes	Yes	Yes
0x1	0x050	File Transfer Begin	No	Yes	Yes
0x1	0x051	File Transfer Begin	No	Yes	Yes
0x1	0x100	Configure ICY-NAME	No	Yes	Yes
0x1	0x101	Configure ICYGENRE	No	Yes	Yes
0x1	0x102	Configure ICYURL	No	Yes	Yes
0x1	0x103	Configure ICYPUB	No	Yes	Yes

```
Message types 0x100 to 0x103 are used to configure properties for a SHOUTcast
1.8 stream, should the distribution point support SHOUTcast listeners.
```

Communication Stages

There are six major phases of broadcaster communication with the distribution point:

- Cipher Key Exchange
- Connect / Authentication
- Stream Configuration
- Intro / Backup File Transfer
- Standby / Data Transfer
- Broadcast Termination

Cipher Key Exchange

In Ultravox 2.1, the UID and Auth-Blob fields of message type 0x1001 (Authenticate Broadcast) are encrypted using the XTEA algorithm. When the broadcaster and server connect, the broadcaster requests the cipher by sending the 0x1009 message. This allows the server to distinguish between Ultravox 2.0 and Ultravox 2.1 (Ultravox 2 begins with the Authenticate message - 0x1001) the server response message contains a null terminated string of a maximum length of 16 bytes (128 bits). This is the cipher key, and is used to encrypt the UID and Auth-Blob fields of message type 0x1001.

Because the UID and Auth-blob fields are colon seperated, we do not use the result of the XTEA algorithm directly. It is possible that the direct output of the XTEA algorithm could have a colon in it, which would cause a parsing error of message 0x1001. Instead we use the result of the XTEA algorithm as a series of 32 bit hex digits.

C++ code for encoding and decoding:

```
// from wikipedia. Slightly modified to be 32/64 bit clean
static void XTEA_encipher(__uint32* v, __uint32* k,unsigned int num_rounds = 32)
{
  __uint32 v0=v[0], v1=v[1], i;
  __uint32 sum=0, delta=0x9E3779B9;
  for(i=0; i<num_rounds; i++)
  {
    v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + k[sum & 3]);
    sum += delta;
    v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + k[(sum>>11) & 3]);
  }
  v[0]=v0; v[1]=v1;
}

static void XTEA_decipher(__uint32* v, __uint32* k,unsigned int num_rounds = 32)
{
  __uint32 v0=v[0], v1=v[1], i;
  __uint32 delta=0x9E3779B9, sum=delta*num_rounds;

  for(i=0; i<num_rounds; i++)
  {
    v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + k[(sum>>11) & 3]);
    sum -= delta;
    v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + k[sum & 3]);
  }
  v[0]=v0; v[1]=v1;
}

static __uint32 fourCharsToLong(__uint8 *s)
{
  __uint32 l = 0;
  l |= s[0]; l <<= 8;
  l |= s[1]; l <<= 8;
  l |= s[2]; l <<= 8;
  l |= s[3];
  return l;
}

static void longToFourChars(__uint32 l,__uint8 *r)
{
  r[3] = l & 0xff; l >>= 8;
  r[2] = l & 0xff; l >>= 8;
  r[1] = l & 0xff; l >>= 8;
  r[0] = l & 0xff; l >>= 8;
}

#define XTEA_KEY_PAD 0
#define XTEA_DATA_PAD 0

static string XTEA_encipher(const __uint8* c_data,size_t c_data_cnt,
                            const __uint8* c_key,size_t c_key_cnt) throw()
{
  ostringstream oss;

  vector<__uint8> key(c_key,c_key + c_key_cnt);
  vector<__uint8> data(c_data,c_data + c_data_cnt);

  // key is always 128 bits
  if(key.size() < 16) key.resize(16, XTEA_KEY_PAD); // pad key with zero
  __uint32 k[4] = { fourCharsToLong(&key[0]),fourCharsToLong(&key[4]),
                    fourCharsToLong(&key[8]),fourCharsToLong(&key[12])
  };

  // data is multiple of 64 bits
  size_t siz = data.size();
  if(siz % 8) { siz+= 8 - siz % 8; data.resize(siz, XTEA_DATA_PAD);} // pad data with zero
```

```
  for(size_t x = 0; x < siz; x+=8)
  {
    __uint32 v[2];
    v[0] = fourCharsToLong(&data[x]);
    v[1] = fourCharsToLong(&data[x+4]);
    XTEA_encipher(v,k);
    oss << setw(8) << setfill('0') << hex << v[0];
    oss << setw(8) << setfill('0') << hex << v[1]; // hex values.
    // uvox uses colon as separator so we can't use chars for fear of collision
  }
  return oss.str();
}

static utf8 XTEA_decipher(const __uint8* c_data,size_t c_data_cnt,
                          const __uint8* c_key,size_t c_key_cnt) throw()
{
  utf8 result;

  vector<__uint8> key(c_key,c_key + c_key_cnt);
  vector<__uint8> data(c_data,c_data + c_data_cnt);

  // key is always 128 bits
  if(key.size() < 16) key.resize(16, XTEA_KEY_PAD); // pad key with zero
  __uint32 k[4] = { fourCharsToLong(&key[0]),fourCharsToLong(&key[4]),
                    fourCharsToLong(&key[8]),fourCharsToLong(&key[12])
  };

  // data is multiple of 16 hex digits
  size_t siz = data.size();
  assert(!(siz % 16)); // should never happen if data is good
  if(siz % 16) { siz+= 8 - siz % 8; data.resize(siz, '0');} // pad data with zero

  for(size_t x = 0; x < siz; x+=16)
  {
    __uint32 v[2];

    sscanf((const char *)&data[x],"%8x",&v[0]);
    sscanf((const char *)&data[x+8],"%8x",&v[1]);
    XTEA_decipher(v,k);
    __uint8 ur[5] = {0,0,0,0,0 };
    longToFourChars(v[0],ur);
    result += ur;
    longToFourChars(v[1],ur);
    result += ur;
  }
  return result;
}
```

Connect / Authentication

Request Cipher

The broadcaster makes a connection to the distribution point and requests the cipher key.

```
Message Class – 0x1
Message Type – 0x009
Payload – <Version><NUL>
<Version> - 2.1.
<NUL> - ASCII 0x00
```

Reponses are:

```
ACK:<cipherkey>\0
```

or:

```
NAK:<Reason>\0
```

Broadcast Authentication Request

The broadcaster makes a connection to the distribution point and must issue an authentication request. The authentication request payload looks like:

```
Message Class – 0x1
Message Type – 0x001
Payload – <Version>:<SID>:<UID>:<AuthBlob><NUL>
<Version> - protocol version number (required, numeric, non-zero, maximum 255).  The current version is 2.1.
<SID> - stream identifier (required, numeric, non-zero, maximum 2147483647).
<UID> - user identifier (required, XTEA encoded).
<AuthBlob> - authentication information (required, XTEA encoded).
<NUL> - ASCII 0x00
```

Since this is the first message required by the distribution point, it contains a version number that can be used by the distribution point to handle future versions of the protocol specification. The distribution point will disconnect the broadcaster if the version is not supported.

The distribution point also has the possibility of using the provided authentication credentials to verify the authenticity of the broadcaster. If the broadcaster is successfully authenticated, the distribution point responds with an Authenticate Broadcast message with a payload that looks like:

```
ACK:<Version>:Allow\0
```

If the broadcaster fails authentication, the distribution point responds with an Authenticate Broadcast message that looks like:

```
NAK:<Version>:<Reason>\0
```

Valid reasons for a failed authentication are as follows:

Reason	Description
“Deny”	Broadcaster denied by authentication service.
“Sequence Error”	The message was received out of sequence.
“Parse Error”	The payload could not be parsed successfully.
“Version Error”	The version is greater than what is supported.
“Stream ID Error”	The stream identifier is out of the supported range (1 - 2147483647).
“Stream Moved”	The stream is configured as having been moved from the server.

Following a NAK response to a Broadcast Authenticate response, the distribution point should terminate the connection.

The completion of the authentication request and response signals moving onto the stream configuration

phase.

Stream Configuration

The stream configuration requests sent by the broadcaster can come in any sequence with the restrictions listed above in 'Broadcast Messages Table'.

Stream Mime Type

This message indicates the mime type of the stream that will be transmitted in this session.

```
Message Class – 0x1
Message Type – 0x040
Payload – <mimetype><NUL>
Possible mime types are
  audio/mpeg
  audio/aacp
  audio/aac
  audio/ogg
<NUL> - ASCII 0x00
```

If the mime type is valid, the distribution point sends a Stream Mime Type message back to the broadcaster with a message payload of “ACK\0” (Length=4). However if the distribution point rejects the broadcaster information it may respond with a Stream Mime Type message with a message payload of “NAK:<Reason>\0”

Setup Broadcast

The setup broadcast message contains details about the stream that the broadcaster will feed through the distribution point. It includes the Content-Type of the stream. It is the responsibility of the distribution point to pass this information to listeners during a listener request for this stream. It also contains the average bitrate and maximum bitrate of the stream, allowing for the possibility of variable bitrate streams. For fixed bitrate streams, the average bitrate will equal the maximum bitrate.

The setup broadcast request looks like:

```
Message Class – 0x1
Message Type – 0x002
Payload – <Avg Bit Rate>:<Max Bit Rate><NUL>
<Avg Bit Rate> – average bit rate (required, numeric, maximum 320, kbps).
<Max Bit Rate> – maximum bit rate (required, numeric, maximum 320, kbps).
<NUL> - ASCII 0x00
```

If the broadcast setup is successfully configured with the given values, the distribution point sends a Broadcast Setup message back to the broadcaster with a message payload of “ACK\0” (Length=4). However if the broadcaster rejects the broadcaster information it may respond with a Broadcast Setup message with a message payload of “NAK:<Reason>\0” (with the corresponding Length) where the reasons are:

Reason	Description
“Sequence Error”	The message was received out of sequence.
“Parse Error”	The payload could not be parsed successfully.
“Bit Rate Error”	The bit rate is not valid or not supported.

Negotiate Buffer Size

Inherently, the distribution point will have some internal buffer for Ultravox Messages that it uses to source listener streams. The Negotiate Buffer Size message allows the broadcaster and distribution point to communicate some properties of that buffering mechanism. The broadcaster can ask the distribution point to setup a buffer of a specific size with a minimum threshold. The distribution point could otherwise have an internal buffer size defined that is too small for the amount of stream data, making it very likely that listeners of the stream will receive a poor experience.

The Negotiate Buffer Size message looks like:

```
Message Class – 0x1
Message Type – 0x003
Payload – <Desired Buffer Size>:<Minimum Buffer Size><NUL>
<Desired Buffer Size> – desired buffer size (required, numeric, in units of KB).
<Minimum Buffer Size> – minimum buffer size (required, numeric, in units of KB).
<NUL> - ASCII 0x00
```

If the distribution point supports a buffer size greater than the minimum buffer size, it responds with a Negotiate Buffer size message with payload “ACK:<negotiated buffer size>\0”. The negotiated buffer size value is in units of KB. However, a failure to be able to provide an adequate buffer size results in the distribution point responding with a Negotiate buffer size message with payload “NAK:<Reason>\0”. Possible reasons are:

Reason	Description
“Sequence Error.”	The message was received out of sequence.
“Parse Error.”	The payload could not be parsed successfully.
“Buffer Size Error.”	The buffer size is not valid or not supported.

```
The return string consists of the trail period ‘.’
```

Configure ICY-NAME

In order to support SHOUTcast 1.8.x audio streams the protocol has adapted to give the distribution point the necessary information. The first of these messages allows the broadcaster the ability to configure the “icy-name” variable. This message is optional and doesn’t apply to distribution points that are not concerned with SHOUTcast 1.8.x compatibility for audio streams. The payload of the Configure ICY-NAME message looks like:

```
Message Class – 0x1
```

```
Message Type – 0x100
Payload – <ICYNAME><NUL>
<ICY-NAME> - value (no limit, utf8 encoded).
<NUL> - ASCII 0x00
```

If the Distribution point successfully sets the value, it replies to the broadcaster by sending the ICY-NAME message with a payload of "ACK\0" (Length=4). Otherwise, it replies by sending the ICY-NAME message with a payload of "NAK:<Reason>\0". Possible reasons for a NAK are:

Reason	Description
"Sequence Error"	The message was not received during the configuration phase.
"Parse Error"	The payload could not be parsed successfully.
"Compatibility mode not enabled"	SHOUTcast compatibility-mode is not enabled.

Configure ICYGENRE

In order to support SHOUTcast 1.8.x audio streams the protocol has adapted to give the distribution point the necessary information. The first of these messages allows the broadcaster the ability to configure the "icy-genre" variable. This message is optional and doesn't apply to distribution points that are not concerned with SHOUTcast 1.8.x compatibility for audio streams. The payload of the Configure ICY-GENRE message looks like:

```
Message Class – 0x1
Message Type – 0x101
Payload – <ICYGENRE><NUL>
<ICY-GENRE> - value (no limit. utf8 encoded).
<NUL> - ASCII 0x00
```

If the distribution point successfully sets the value, it replies to the broadcaster by sending the Configure ICYGENRE message with a payload of "ACK\0" (Length=4). Otherwise, it replies by sending the Configure ICYGENRE message with a payload of "NAK:<Reason>\0". Possible reasons for a NAK are:

Reason	Description
"Sequence Error"	The message was not received during the configuration phase.
"Parse Error"	The payload could not be parsed successfully.
"Compatibility mode not enabled"	SHOUTcast compatibility-mode is not enabled.

Configure ICYURL

In order to support SHOUTcast 1.8.x audio streams the protocol has adapted to give the distribution point the necessary information. The first of these messages allows the broadcaster the ability to configure the "icy-url" variable. This message is optional and doesn't apply to distribution points that are not concerned

with SHOUTcast 1.8.x compatibility for audio streams. The payload of the Configure ICY-URL message looks like:

```
Message Class – 0x1
Message Type – 0x102
Payload – <ICYURL><NUL>
<ICY-URL> - value (no limit, utf8 encoded).
<NUL> - ASCII 0x00
```

If the distribution point successfully sets the value, it replies to the broadcaster by sending the Configure ICYURL message with a payload of “ACK\0” (Length=4). Otherwise, it replies by sending the Configure ICYURL message with a payload of “NAK:<Reason>\0” (with the corresponding Length):

Reason	Description
“Sequence Error”	The message was not received during the configuration phase.
“Parse Error”	The payload could not be parsed successfully.
“Compatibility mode not enabled”	SHOUTcast compatibility-mode is not enabled.

Configure ICYPUB

In order to support SHOUTcast 1.8.x audio streams the protocol has adapted to give the distribution point the necessary information. The first of these messages allows the broadcaster the ability to configure the “icy-pub” variable. This message is optional and doesn’t apply to distribution points that are not concerned with SHOUTcast 1.8.x compatibility for audio streams. The payload of the Configure ICY-PUB message looks like:

```
Message Class – 0x1
Message Type – 0x103
Payload – <ICYPUB><NUL>
<ICY-PUB> - value (required, numeric, either 0 or 1).
<NUL> - ASCII 0x00
```

If the distribution point successfully sets the value, it replies to the broadcaster by sending the Configure ICYPUB message with a payload of “ACK\0” (Length=4). Otherwise, it replies by sending the Configure ICYPUB message with a payload of “NAK:<Reason>\0” (with the corresponding Length):

Reason	Description
“Sequence Error”	The message was not received during the configuration phase.
“Parse Error”	The payload could not be parsed successfully.
“Compatibility mode not enabled”	SHOUTcast compatibility-mode is not enabled.

Intro / Backup File Transfer

The distribution point can contain special files. "Intro" files are played whenever a listener connects to the station. "Backup" files are played if contact with the broadcaster is lost. These files can be transfered to the distribution point using the File Transfer Begin / Data messages. The mime-type and bitrate for the file must match the configuration for streaming. These messages may appear at any point once streaming begins.

File Transfer Begin

When the broadcaster is ready to send one of the special files, it starts by sending the File transfer begin message, which indicates what type of file is being sent.

```
Message Class - 0x1
Message Type - 0x050
Payload - <type>:<size in bytes>NUL
  Type is one of:
    intro
    backup
Size in bytes is the number of bytes in the file that is to be transmitted.
```

The distribution point responds with ACK\0 or NAK:<reason>\0 If a File transfer begin message is received before all the data for a prior begin has been received, then the prior transfer is considered corrupt and should be ignored.

File Transfer Data

One or more of these messages follow a successful File transfer begin message. The payload of the message is the data of the file being transfered. More than one of these messages are required if the file contents do not fit in the payload limits negotiated during the stream configuration phase.

```
Message Class - 0x1
Message Type - 0x051
Payload - binary data
```

The distribution does not send a response to this message

Standby / Data transfer

Standby

When the broadcaster is done sending configuration requests and is ready to begin streaming, it sends Standby message to the distribution point. This message has no payload and looks like:

```
Message Class – 0x1
Message Type – 0x004
```

```
Payload - <empty>
```

It is the responsibility of the distribution point to make sure all required configuration and authentication messages have been processed successfully. If the distribution point has completed setup successfully and is ready to accept stream data from the broadcaster, it responds with a Standby message with payload "ACK:Data transfer mode\0". If the setup of the broadcaster has not been completed or there is a problem the distribution point responds "NAK:<Reason>\0". Possible reasons are:

Reason	Description
"Sequence Error"	The message was received out of sequence.
"Parse Error"	The payload could not be parsed successfully.
"Configuration Error"	The configuration is not complete.
"Stream In Use"	The distribution point cannot accept the broadcaster.

Flush Cached Metadata

Information associated with a stream may be passed in metadata messages through the distribution point. Some of these messages may be cached in the distribution point. This message informs the distribution point to throw out existing cached metadata messages. This message has no payload and looks like:

```
Message Class – 0x1
Message Type – 0x006
Payload - <empty>
```

The distribution point responds with a Flush Cached Metadata message with payload "ACK\0" or if there is a problem the distribution point responds "NAK:<Reason>\0".

Please refer to the Metadata Message section for full details on how the distribution point will handle metadata.

Require Listener Authentication

The disctribution point and the broadcaster need to agree on a maximum payload size. This message allows the broadcaster to specify it's desired Max message and a lower-limit:

```
Message Class – 0x1
Message Type – 0x007
Payload - <Ultravox Requirement >:<SHOUTcast Requirement ><NULL>
<Ultravox Requirement > – (required, character, 'Y', 'N', or 'D').
<SHOUTcast Requirement> – (required, character, 'Y', 'N', or 'D').
<NUL> - ASCII 0x00
```

This request provides the broadcaster with the ability to specify an authentication requirement different from the configured default. If the distribution point successfully sets the authentication requirements, it replies to the broadcaster by sending the Error! Reference source not found. message with a payload of "ACK\0". Otherwise, it replies by sending the Error! Reference source not found. message with a payload of "NAK:<Reason>\0" (with the corresponding Length):

Reason	Description
“Sequence Error”	The message was received out of sequence.
“Parse Error”	The payload could not be parsed successfully.

Negotiate Max Payload Size

The distribution point and the broadcaster need to agree on a maximum payload size. This message allows the broadcaster to specify it’s desired Max message and a lower-limit:

```
Message Class – 0x1
Message Type – 0x008
Payload - <desired max-payload size>:<minimum acceptable max-payload size><NUL>
```

If the Max Payload Size negotiation is successfully configured with the given values, the distribution point sends a Broadcast Setup message back to the broadcaster with a message payload of “ACK:<negotiated max-payload size>\0”.

However if the broadcaster rejects the broadcaster information it may respond with a Negotiate Max Payload Size message with a message payload of “NAK:<Reason>\0” (with the corresponding Length) where the reasons are:

Reason	Description
“Sequence Error”	The message was received out of sequence.
“Parse Error”	The payload could not be parsed successfully.
“Payload Size Error”	The minimum acceptable max-payload size is not supported

The maximum acceptable payload size is 16377 bytes. The value is derived from 16 * 1024 – 6 – 1, which translated to 16k minus 6, the number of header bytes and 1, the null terminator byte.

Broadcast Termination

Terminate Broadcast Message

During Data Transfer Mode the broadcaster may decide to close the connection to the distribution point. For clean handling of a disconnect the broadcaster will send a Terminate Broadcast message. The distribution point will not expect any messages following a Terminate Broadcast message and will unsubscribe any listeners and terminate the stream, freeing its resources. The message looks like:

```
Message Class – 0x1
Message Type – 0x005
Payload - <empty>
```

Listener Messages

Listener messages are messages originating from the distribution point and received by the listener. These messages are often informative and the listener does not send responses. The messages of this class do not have any payload, however it is possible that future messages in this class could.

Temporary Broadcast Interruption

There are other circumstances that may cause the broadcast to become unavailable. The distribution point may detect these occurrences. If broadcast unavailability is detected, the distribution point can send a Temporary Broadcast Interruption message to the listener while it attempts to recover the broadcast. The Temporary Broadcast Interruption message looks like:

```
Message Class – 0x2
Message Type – 0x001
Payload - <empty>
```

Broadcast Termination

This message is sent to a listener when the stream they are subscribed is terminating. The distribution point sends the message to all listeners subscribed to the terminating stream and then closes the connection. The broadcast termination message looks like:

```
Message Class – 0x2
Message Type – 0x002
Payload - <empty>
```

Broadcast Failover

When a listener fails over to a new stream, it will receive an indication that they have failed over. This indication includes the operating parameters of the new stream, and the cacheable metadata from the new stream as if they have just joined.

Here is the sent data after a broadcast failover:

- An indication that a broadcast fail condition has occurred (0x2003)
- Stream parameters:

```
Avg BPS - As the new broadcaster may be broadcasting at a different rate.
Max BPS – As the new broadcaster may be broadcasting at a different rate.
Ultravox-Max-Msg value - Since this is used for reset detection and recovery, and is
```

```
set by the broadcaster; this can change from broadcaster to broadcaster.
```

- The next message follow should be a cacheable metadata as if it were joining a new stream.

```
Message Class – 0x2
Message Type – 0x003
Payload - <New SID>:music/ultravox:<Avg BPS>:<Max BPS>:<Ultravox-Max-Msg>
```

Metadata

Metadata is any information associated with a stream to be interpreted by the listener. Examples of metadata include Stream-Title, Stream-URL (web-site content related to the streamed content), Stream-Image(binary data), etc. Metadata information should flow over the same data path as stream data. The reason for this is that web proxies and firewalls often prevent for easy management of multiple data connections. It is the responsibility of the broadcaster not to saturate a stream with too much metadata. Doing so will create a poor listener experience that could be avoided by intelligent management of metadata.

Metadata originates from a broadcaster and is passed through the distribution point for handling by the listener. It is sent relatively rarely in comparison to the stream data. For example, an audio stream might have a metadata message that indicates a change in the song title. If a listener subscribes to the stream just after the song-change metadata has passed, the listener won't know the name of the song they are listening to until the next song change. Therefore the protocol distinguishes two types of metadata.

Cachable metadata is vital to the stream and the listener should always have the most recent cachable metadata. Pass-thru metadata is stream-associated data that is nice to have and that the listener won't mind waiting for.

Cachable Metadata

Cachable metadata requires the distribution point to cache metadata messages and pass them down to the listener upon listener connects. Therefore, the point in the stream buffer at which data is cached should coincide with where a listener is placed in the stream buffer on a listener connect. Cached metadata messages are also included in the data stream to the listener during normal streaming. This ensures that listeners have the most recent cached metadata both at the beginning of a streaming session and at all times afterward. Metadata messages looks like:

```
Message Class – 0x3 or 0x4
Message Type - <metadata type>
Payload – [Metadata ID][Metadata Span][Metadata Index][metadata]
[Metadata ID] (16bits) - Used to identify a metadata set for when metadata is split across multiple Ultravox messages
[Metadata Span] (16 bits) - the number of messages comprising the complete metadata package (numeric, minimum 1, maxim
[Metadata Index] (16 bits) - the ordinal identification of this message within the metadata package (numeric, minimum
[Metadata] – the metadata information.
```

As you can see from the sequencing system, metadata may be split into multiple messages. Multiple metadata messages of the same type and metadata ID can be combined and interpreted by a listener as a single piece of metadata. Similarly the distribution point will need to understand this in order to properly cache fragmented metadata. If for a given message type, a message is received whose index is already cached, the distribution point should remove all the messages it has cached for that message type and then

caches the new message.

Pass-thru Metadata

These messages take the same form as Cachable metadata with message classes of either 0x5 or 0x6. However, the distribution point simply treats these messages like data messages and passes them directly to the listeners.

Broadcast Connection Handling

The following connection handling support is recommended:

Normal Termination

Normally, the broadcaster stops providing a stream by sending the Terminate message to the distribution point, thereby making the stream no longer available for subscription by listeners. All other circumstances (specifically idle-timeout and reconnect-timeout) that result in the termination of a stream are considered abnormal.

When a stream terminates normally, the distribution point informs all listeners currently subscribed to the stream that the stream is terminated by sending them a Broadcast Terminate message.

Idle Timeout

If the distribution point does not receive either a data or metadata message while in Data Streaming mode for some time, it may enforce an Idle Timeout. Idle timeout occurs and the distribution point disconnects the broadcaster and abnormally terminates the stream it was providing. This is a recommendation for connection handling and not a requirement for protocol compatability.

Reconnect Timeout

If the TCP connection to a broadcaster is unexpectedly lost, the distribution point may allow the broadcaster an interval of time to reconnect and re-establish the stream. Any number of reasons might cause this to happen including network glitch, software error, etc. The broadcaster has the option to re-establish the stream by sending the Authenticate Broadcast (0x1001) message. However if the distribution point detects that the broadcaster is reconnecting, it can choose to respond with an ACK message for the Standby(0x1004) message. The configuration phase will be skipped and the broadcaster can resume streaming in data transfer mode. The distribution point should only send 0x1004 if the previous broadcast had completed its configuration phase.

If the broadcaster does not reconnect and re-establish the stream, reconnect timeout occurs and the

distribution point abnormally terminates the stream the broadcaster was providing. This is a suggested feature for the distribution point but should be handled by broadcasters.

Ultravox Listener Protocol

The Ultravox Listener Protocol is the handshaking necessary for an Ultravox listener to connect, setup, and terminate a streaming session. The protocol is based on the HTTP protocol to allow listeners behind firewalls and proxies to make a request without additional configuration or security clearance. The Ultravox listener represents a new type of software that can interpret and properly handle Ultravox messages. Ultravox listeners are recognized by the presence of the sub-string “ultravox/1.0” in the “UserAgent” header of the initial HTTP GET request.

Listen Request

As part of the protocol, a listener supplies the SID to identify the stream of interest to the distribution point. An Ultravox Listen request looks like:

```
GET /{path} HTTP/1.0\r\n
Host: ultravox.aol.com\r\n
UserAgent: Ultravox/2.1\r\n
Accept: */*\r\n
…
\r\n
```

The user agent is important. The version of sc_serv that supports Shoutcast 2.0, looks for the string "Ultravox/2.1" in the user agent. If the string is found anywhere in the user agent, then Shoutcast 2.0 is assumed, otherwise the server will downgrade to Shoutcast 1.

If the distribution point accepts the connection, authenticates the listener, and sets up a streaming session for the listener, it responds with:

```
HTTP/1.1 200 OK\r\n
Server:Ultravox/2.1 SHOUTcast v2.0.0.29\r\n
ContentType:misc/ultravox\r\n
icy-pub:<public flag>\r\n
Ultravox-Bitrate:<stream bitrate>\r\n
Ultravox-Title:<stream title>\r\n
Ultravox-Genre:<stream genre>\r\n
Ultravox-URL:<broadcaster url>\r\n
Ultravox-Max-Msg:<negotiated max payload size>\r\n
Ultravox-Class-Type:<class-type>\r\n
```

where the “negotiated max payload size”, “stream title”, “stream bitrate”, and “class-type” are populated based on the values provided during broadcast negotiation.

“server” returns at least 'Ultravox/2.1' but usually also includes the version of the server or any other information which is deemed of use to provide to the listener.

class-type is a hexadecimal representation like “7000” for an MP3 (audio/mpeg) stream.

The following are a list of possible responses for a Listen connection:

Response	Description
“HTTP/1.1 200 OK”	Authentication allowed
“HTTP/1.1 400 Bad Request”	Request can not be parsed
“HTTP/1.1 403 Forbidden”	Authentication denied
“HTTP/1.1 404 Not Found”	Stream not available

Prebuffering

Prebuffering is the mechanism by which the listener is ensured a smooth streaming experience. Since network conditions may change quickly and fluctuate over the duration of a streaming session, the distribution point sends some data to the listener at the beginning of the streaming session to be used to pad the listening experience. Stream data is then inserted behind the buffered data while the listener reads data from the front of the buffer. To setup the listen session more quickly, the prebuffer data can be sent at an accelerated rate. However, the distribution point should respect the listener’s TCP window when sending at an accelerated rate and make sure to account for TCP slow start. The amount of data sent in the prebuffer phase is determined by the number of seconds specified by the prebuffer time argument in the HTTP GET request. If the prebuffertime is not provided the distribution point will make an assumption about the prebuffersize. A prebuffer time of zero means that no data is prebuffered.

Resetting

If the listener is not able to keep up with the bitrate of the stream, it falls behind. If the listener falls far behind, the distribution point must overwrite stream data that the listener has not yet received. It is at this time that the distribution point must reset the listener, skipping them ahead in the stream data.

It is possible (depending on implementation) that Ultravox message boundaries could be corrupted while moving the listener’s pointer inside of the buffer such that the listener receives corrupt data on a reset. The distribution point is not required to abide by message boundaries when moving the listener, however it’s recommended.

The mechanism for resynchronization is handled entirely by the client based on the following:

- Ultravox-Max-Msg value
- Message length
- Trailing 0x00 byte
- 5A sync byte

The methodology is as follows:

1. If message starts with 0x5A, read payload length.
2. If length is greater than Ultravox-Max-Msg, then search for next 5A sync byte and start over.
3. Read in enough bytes to check the trailer 0x00 after the payload
4. If 0x00 is not present, search from the beginning of the bogus message to the next 5A and start over.
5. Otherwise, extract the payload and continue reading and extracting ultravox messages.

Known Message Types

The following is the listing of known messages (split into class and type) which are either in active usage with currently implemented versions of the protocol or have been specified as reserved to enable future usage or expansion of the protocol usage.

Class	Type	Description
0x0		Operations
0x1		Broadcaster
0x1	0x001	Authenticate Broadcast
0x1	0x002	Setup Broadcast
0x1	0x003	Negotiate Buffer Size
0x1	0x004	Standby
0x1	0x005	Terminate
0x1	0x006	Flush Cached Metadata
0x1	0x007	Require Listener Auth
0x1	0x008	Negotiate Max Payload Size
0x1	0x009	Request Cipher
0x1	0x040	Stream mime type
0x1	0x050	File transfer begin
0x1	0x051	File transfer data
0x1	0x100	Configure ICY-NAME
0x1	0x101	Configure ICYGENRE
0x1	0x102	Configure ICYURL
0x1	0x103	Configure ICYPUB
0x2		Listener
0x2	0x001	Temporary Broadcast Interruption
0x2	0x002	Broadcast Termination
0x3		Cacheable Metadata
0x3	0x000	Content Info Metadata (unused)
0x3	0x001	Url Metadata (unused)
0x3	0x901	XML Metadata (Aol Radio format)
0x3	0x902	XML Metadata (SHOUTcast 2.0 format)
0x4		Cacheable Binary Metadata (reserved)
0x4	0x0xx	Station logo (see 'Image Notes' for xx details)
0x4	0x1xx	Album art (see 'Image Notes' for xx details)
0x5		Pass-through Metadata
0x5	0x001	Time Remaining (reserved)
0x6		Pass-through Metadata (unused)
0x7	0x000	Data - MP3
0x8	0x000	Data - VLB
0x8	0x001	Data - AAC LC
0x8	0x003	Data - AACP
0x8	0x004	Data - Vorbis (reserved)
0x9	0x000	Data - Headerless AACP (reserved) (see 'Framed Data' for details)

0x9	0x001	Data - Headerless Vorbis (reserved) (see 'Framed Data' for details)
0xA		Cacheable Binary Metadata (reserved for use with 0x9 if implemented)

Image Notes

When these are received, the xx of the type is used to specify the image mime type where:

00	image/jpeg
01	image/png
02	image/bmp
03	image/gif

Framed Data

This is a potential extension intended to provide encoded data using just the Ultravox frame instead of the usual overhead required when streaming existing formats. The aim of this if implemented (currently a proposed feature) is it can help reduce the overhead of streaming at lower bitrates where this is more noticeable e.g. the potential of saving 7 bytes per audio frame for AAC streaming. However this would also require the listener to be compatible and be able to cope otherwise the distribution point would need to convert the frames back to ones containing a header.

Conclusion

Using the Ultravox Broadcast and Listener Protocol will make it possible stream data of different types (audio, video, animation codecs) to an end user. This protocol gives specific information for building these services as well as recommendations for managing the system. The fact that the broadcaster has complete control over the desired bitrate makes this possible along with the logic to generalize a stream in the distribution point.

Glossary

Authentication Server – a service that the distribution point may use to authenticate broadcasters and listeners when they initiate a connection to the distribution point.

Auth-Blob – a character string (ASCIIZ), no longer than 1200 bytes, sent to an authentication server with a UID as authentication credentials.

Broadcaster – a process that implements the Ultravox Broadcaster Protocol of the distribution point providing a media stream using the Ultravox Broadcaster Protocol across a connection.

Idle-timeout – the maximum time interval that the Distribution point allows between messages received from a broadcaster before it disconnects the broadcaster and considers the stream it was providing to be abnormally terminated.

Listener – a network client of the Distribution point subscribing to a media stream using either the Ultravox Listener Protocol or the SHOUTcast Listener Protocol across a TCP connection.

Media – the encoded audio/video data (e.g., MP3) which is the essential content of a stream. Media is "streamed" from the broadcaster to be "played" by a listener.

Metadata – data associated with a stream that is not media, but contains information that is pertinent to the media being provided (e.g., titles, advertisements, graphics, etc…).

Prebuffering - the process by which the Distribution point fills the listener's stream data buffer as quickly as possible in order allow a continuous playback even when there are minor variations in network latency.

Reconnect-timeout – the maximum time interval that the Distribution point allows a broadcaster to reconnect and re-establish a stream when a broadcaster unexpectedly disconnects.

SHOUTcast – the streaming media protocol used by an installed base of broadcasters and listeners.

SID (Stream IDentifier) – a non-zero integer (32-bit unsigned) value that uniquely identifies a particular media stream within a Distribution point. The association of SIDs with streams is managed operationally outside of the Distribution point.

Stream – a continuous sequence of media and associated metadata provided to the Distribution point by a broadcaster.

Termination – the condition of a stream whereby it is no longer available for subscription by listeners. Normal termination of a stream occurs when the broadcaster providing it makes a specific request to terminate. All other circumstances (specifically idle-timeout and reconnect-timeout) that result in the termination of a stream are considered abnormal.

UID (User IDentifier) – a character string (ASCIIZ), no longer than 64 bytes, that identifies a particular broadcaster or listener for the purpose of authentication.

Ultravox Protocol - the streaming media protocol(s) intended to supercede the SHOUTcast protocol.

Retrieved from "http://wiki.shoutcast.com
/index.php?title=SHOUTcast_2_(Ultravox_2.1)_Protocol_Details&oldid=74446"

- This page was last modified on 28 October 2014, at 22:57.