SlimProto TCP protocol

From SqueezeboxWiki

note always check the perl source code, this documentation is not in sync with actual byte length in multiple places

Contents

SlimProto Protocol

The SlimProto protocol is the new protocol developed for the squeezebox and replaces the SLIMP3 one. It is designed to allow the players to communicate effectively over WANs as well as LANs.

This document describes the SlimProto for Squeezebox firmware versions 20 and above. The current version of the firmware shipped with SqueezeCenter 7.2.1 is 113. This document describes both version imperfectly.

The server listens on TCP port 3483 for connections by players. To register a player with the server, they exchange "helo"s and then any of the commands below are valid.

The client also listens on UDP port 3483 for SlimProto commands from the server it has already established a TCP connection with. It also seems to respond to data sent back in the TCP connection (you send frames back this way).

The byte order is critical for several of the fields. The best way to check this out is to look at the server source code.

Client -> Server Communications

For an authoritative answer on the format of the packets see the file Slim/Networking/Slimproto.pm . Different versions of the firmware send different formatted packets.

A command to the server consists of three parts:

1. The 1st 4 bytes specify the operation. The following operations are supported:

- HELO
- BYE!
- STAT heartbeat and play-event notification
- RESP HTTP headers (from data stream)
- BODY HTTP body (from data stream)
- META metadata from stream
- DSCO data stream disconnected
- DBUG reports firmware revision
- IR (note the two spaces after IR)
- RAWI raw infra-red
- ANIC animation complete
- BUTN hardware button (transporter)
- KNOB hardware knob (transporter)
- SETD update preferences
- UREQ firmware update request

2. The 2nd part (of four bytes) is simply the length of the data packet (in network order).

3. The 3rd part is the data itself.

Most of the operations above are detailed below:

HELO

This alerts the server to a clients presence and is the first thing normally transmitted by a client powering up.

Data Length: 10, 20, or 36 bytes, depending upon deviceID and firmware revision.

Format:

Field	Length	Notes
DeviceID	1 byte	The Device ID of the player. '2' is squeezebox. '3' is softsqueeze, '4' is squeezebox2,. '5' is transporter, '6' is softsqueeze3, '7' is receiver, '8' is squeezeslave, '9' is controller, '10' is boom, '11' is softboom, '12' is squeezeplay
Revision	1 byte	A number specifying the firmware revision
MAC 0-5	6 bytes	The player's MAC address
UUID 0-15	16 bytes	Unique identifier of device (only in newer firmware)(not sent by softsqueeze)
WLanChannelList	2 bytes	A list of 802.11 channels that are enabled on a given device as a bitfield 0x07ff means that channels 0 through 11 are enabled which means it's configured for the US.
Bytes received	8 bytes	Number of data-stream bytes received (not sent by softsqueeze)
Language	2 bytes	Country code (not sent by softsqueeze)

Capabilities

This is an extension implemented in SBS 7.4 or later.

The basic HELO packet above can be augmented by a list of capabilities, in which case the packet will be longer than 36 bytes. Capabilities form a comma-separated list. By convention CODEC capabilities are three or four letters all lower case and other capabilities start with an upper-case letter.

Unrecognized capabilities will probably be ignored.

CODECs should be stated in order of preference. The following are recognized at the time of writing:

- **wma**
- **wmap** - WMAPro
- **wmal** - WMA Lossless
- **ogg**
- **flc**
- **pcm**
- **aif**
- **mp3**
- **alc** - Apple Lossless
- **aac** - AAC & HE-AAC, with or without MP4 wrapper

The following could be anticipated:

- **rtsp**
- **aud**

The following other capabilities are supported (case sensitive):

- **MaxSampleRate**=*96000* - frames/s, should be multiples of 11025 or 12000
- **Model**=*controller*
- **ModelName**=*Controller*
- **Rhap** - supports Rhapsody
- **AccuratePlayPoints** - indicates that playpoints are precise and can be used in synchronization calculations without further filtering.
- **SyncgroupID**=*nnnnnnnnnn* - the sync-group which the player wants to join; used when a group of players is moved from one server to another.
- **HasDigitalOut** - somewhat of a misnomer. Currently, its only use is to enable the setting (via the SqueezeboxServer WebUI or CLI) of a preference to tie output volume at 100%.
- **HasPreAmp** - does the player have a separate analogue volume control.
- **HasDisableDac** - can the player's DAC be disabled.

"IR " (Note: padded with two spaces to make it up to 4 characters.)

One of these packets is received for each IR code received by the player.

Data Length: Fixed at 10 bytes.

Format:

```
Time   4 bytes Time since player startup in ticks (@1kHz)
Format 1 byte  Code Format (ignored by the server for now.
               Code represents type of IR code - NEC, JVC or Sony)
NoBits 1 byte  Length of IR Code.
               (ignored by the server for now
               16 bits for JVC, 32 bits for NEC?)
IRCode 4 bytes the IR Code itself (upto 32 bits)
```

RESP

BODY

HTTP (or MMS) response headers or message body from the data stream. These can be parsed for metadata (title, format, bitrate, etc.), or (in the case of a body) for a playlist.

META

Embedded metadata from the data stream from which useful information may be extracted,

STAT

These are sent by the player in response to commands and periodically when there are no commands as a sort of keep-alive. They inform the server of the status of various player internals.

Format:

Size	Notes
u32	Event Code (a 4 byte string)
u8	Number of consecutive CRLF recieved while parsing headers
u8	MAS Initalized - 'm' or 'p'
u8	MAS Mode - serdes mode?
u32	buffer size - in bytes, of the player's (network/stream) buffer

u32	fullness - data bytes in the player's (network/stream) buffer
u64	Bytes Recieved
u16	Wireless Signal Strength (0-100 - Larger values mean hardwired)
u32	jiffies - a timestamp from the player (@1kHz)
u32	output buffer size - the decoded audio data buffer size
u32	output buffer fullness - bytes in the decoded audio data buffer
u32	elapsed seconds - of the current stream
u16	voltage
u32	elapsed milliseconds - of the current stream
u32	server timestamp - reflected from an **strm-t** command
u16	error code - used with **STMn**

The Bytes Received field is how many data bytes the player has recieved (since it started playing?).

Output buffer size/fullness, elapsed time, voltage, elapsed milliseconds, server timestamp & error code, along with the **STMo** were added with more-recent firmware revisions.

Earlier firmware revisions report the read and write pointers for the player's decoder input buffer, rather than the buffer size and fullness. The bytes received and the read/write pointers are used in this case to update the elapsed time count.

Event code may be one of the following:

EventCode	Description	Notes
vfdc	vfd received	Not observed, presumably an ack to a VFD (display) message
i2cc	i2c confirmation	
STMa	Autostart	Track started. Probably Squeezebox v1 only.
STMc	Connect	**srtm-s** command received. Guaranteed to be the first response to an strm-s.
STMd	Decoder ready	Instruct server that we are ready for the next track (if any). Sometimes (unfortunately) this will also indicate an error.
STMe	Stream connection Established	
STMf	Connection flushed	Streaming track flushed (in response to **strm-f**) or playback stopped (in response to **strm-q**). The number of STMf responses which may be received in various circumstances is not well defined.
STMh	HTTP headers received	from the streaming connection.
STMl	Buffer threshold reached	When strm-s *autostart*=0/2.
STMn	Not Supported	Decoder does not support file format or a decoding error has occurred. May include format-specific error code.
STMo	Output Underrun	No more decoded (uncompressed) data to play; triggers rebuffering.
STMp	Pause	Confirmation of Pause.
STMr	Resume	Confirmation of resume. (Not used during playout)
STMs	Track Started	Playback of a new track has started.
STMt	Timer	A simple hearbeat, can be periodic or a response to **strm-t**
STMu	Underrun	Normal end of playback.

The event code is a method of triggering events in the server. For example, **STMd** which occurs on decoder buffer underrun causes the server to switch to the next track.

The error codes are internal and codec-specific. Not all decoders make use of error codes. The current ones are:

Code	Name
0	N/A
100	OGG_ERROR_TOO_MANY_CHANNELS
101	OGG_ERROR_HIGH_MEMORY
102	OGG_ERROR_INVALID_SAMPLE_RATE
103	OGG_ERROR_NOTVORBIS
104	OGG_ERROR_BADHEADER
200	WMA_ERROR_SPEECH_SUPERFRAMES
201	WMA_ERROR_LOSSLESS

ANIC

The display animation is complete. This seems to be undocumented. in the squeezecenter code Slim\Display\Display.pm there a brief description of the client capabilities.

BYE!

The player is disconnecting.

If the first data byte to this command is 0x01 then the player is going out for an upgrade...

Server -> Client Communication

Data from the server to the player consists of 2 bytes of length data (in network order), a 4 byte command header, then the command data itself. The length sent is the size of the data plus the 4 byte command header; it does not include the 2 byte length field itself.

Command: "strm"

This takes 24 bytes data of the form:

Field	Length	Notes
$command	1 byte	's' start, 'p' pause, 'u' unpause, 'q' stop, 't' status, 'f' flush, 'a' skip-ahead
$autostart	1 byte	'0' = don't auto-start, '1' = auto-start, '2' = direct streaming, '3' = direct+auto
$formatbyte	1 byte	'p' = PCM, 'm' = MP3, 'f' = FLAC, 'w' = WMA, 'o' = Ogg., 'a' = AAC (& HE-AAC), 'l' = ALAC
$pcmsamplesize	1 byte	'0' = 8, '1' = 16, '2' = 20, '3' = 32; usually '1', '?' for self-describing formats.
$pcmsamplerate	1 byte	'0'=11kHz, '1'=22kHz, '2'=32kHz, '3'=44.1kHz, '4'=48kHz, '5'=8kHz, '6'=12kHz, '7'=16kHz, '8'=24kHz, '9'=96kHz; usually 3, '?' for self-describing formats.
$pcmchannels	1 byte	'1'=mono, '2'=stereo; usually '2', '?' for self-describing formats.
$pcmendian	1 byte	'0' = big, '1' = little; '1' for WAV, '0' for AIFF, '?' for self-describing formats.
$threshold	1 byte	KB of input buffer data before autostart or notify
$spdif_enable	1 byte	'0'=auto, '1'=on, '2'=off; usually 0
$trans_period	1 byte	transition duration in seconds
$trans_type	1 byte	'0' = none, '1' = crossfade,'2' = fade in, '3' = fade out, '4' = fade in & fade out
$flags	1 byte	0x80 - loop infinitely, 0x40 - stream without restarting decoder, 0x01 - polarity inversion left, 0x02 - polarity inversion right
$output_threshold.	1 byte	amount of output buffer data before playback starts, in tenths of second
RESERVED	1 byte	reserved
$replay_gain	4 bytes	replay gain in 16.16 fixed point, 0 = none (for the 's' command - see below for further discussion).
$server_port	2 bytes	Server Port to use (9000 is the default)
$server_ip	4 bytes	0 means use IP of control server

This is followed by an HTTP header itself. This is used to obtain the stream data eg:

```
GET /stream.mp3?player=$client-id HTTP/1.0
(Authorization: Basic $password)
(blank line)
```

The Auth line is only sent if authorization is in use. $client-id is the usually the MAC address of the player and $password is a password generated by the server.

Other headers to be included in the HTTP request may also be supplied.

u, p, a & t commands and *replay_gain* field

The **u**, **p**, **a** & **t** commands all make special use of the *replay_gain* field, as follows:

u - if non-zero, the player-specific internal timestamp (ms) at which to unpause (this is used by the server to coordinate the start of multiple synchronized players).

p - if non-zero, an interval (ms) to pause for and then automatically resume - no **STMp** & **STMr** status messages are sent in this case.

a - if non-zero, an interval (ms) to skip over (not play).

t - a timestamp field from the server to be returned in the corresponding **STMt** status message (used to measure round-trip latency).

WMA-specific notes

For WMA, some of the fields are reused, as follows:

pcmsamplesize - response will include HTTP/MMS chunking;

pcmsamplerate - the audio stream number to play;

pcmchannels - the metadata stream number, if any.

AAC-specific notes

For AAC, some of the fields are reused, as follows:

pcmsamplesize - container type and bitstream format: '1' (adif), '2' (adts), '3' (latm within loas), '4' (rawpkts), '5' (mp4ff), '6' (latm within rawpkts);

Command: "aude"

Tell the client to enable/disable the audio outputs.

2 bytes

```
$spdif_enable   1 byte  0x0 = disable SPDIF, 0x1 = enable SPDIF
$dac_enable     1 byte  0x0 = disable DAC, 0x1 = enable DAC output
```

Command: "audg"

Tell the client to adjust the audio gain (volume level)

10 bytes (all ints are network byte order)

```
$old_left      4 bytes unsigned int
$old_right     4 bytes unsigned int
$dvc           1 byte  Digital volume control 0/1
$preamp        1 byte  Preamp (byte 255-0)
$new_left      4 bytes 16.16 fixed point
$new_right     4 bytes 16.16 fixed point
$sequence      4 bytes unsigned int, optional
```

total: 18-22 bytes (+ 4 bytes command header "audg")

```
old_left/old_right should range from 0..128 new_left/new_right are 16.16 fixed point
```

```
Firmware v22+ on the SB2 use the new style; older use the old-style.
```

Command: "grfb"

Tells the client to adjust the brightness of the display.

Send a short int (2 bytes, network byte order) with a brightness level from 0-4 (on Squeezebox2). On newer firmwares (squeezecenter 7.3), it seems to range from -1 up to and including 5. -1 means totally off, 5 is the brightest.

Command: "grfe"

Sends a bitmap to the client for display. It starts with a header of 4 bytes, followed by 1280 bytes of data. The header is:

```
$offset        2 bytes short int
$transition    1 byte ('L','R','U' or 'D')
$param         1 byte
```

On Squeezebox3, the data is 1280 bytes; each one is a bitfield (0=off 1=on for that bit). Bits are layed out from top to bottom in column 1 of the display, then top to bottom in column 2, etc. The Squeezebox3 has a 320x32 display, so each column is 4 bytes.

if $transition is a capital, the frame is bounced a few pixels, a lowercase value results in a full scroll. the parameter determines how many pixels are part of the animation (from bottom to top). More information on the transition can be found in pushBumpAnimate() of slim\display\Squeezebox2.pm in the squeezecenter source code.

For instance, the 4-byte sequence (in binary):

```
1000 0000
0000 0000
0000 0000
0000 0011
```

would light up the top pixel and the two bottom pixels on the row; send a grfe with that sequence 320 times and you get a thin line across the top of the display and a thicker line across the bottom.

It is also possible to send compressed graphics. the highest byte of 'g' in the 'grfe' command should be set. The frame is then assumed to be compressed using LZF. The length in the header is the length of the compressed data (+4 for the command code).

Command: "i2cc"

Squeezebox1 only, sends an i2c command to the client.

The data is the i2c command to send in the same format as for the SLIMP3 protocol.

This is used to control the setup of the mas chip. Volume commands are embedded in this.

Command: "serv"

Tells the client to switch to another server.

```
$ip_address   4 bytes in network order; 0x1 means switch to squeezenetwork
$syncgroupid  (optional) 10 ASCII digits;
              this should be reflected in the HELO command
              (SyncgroupID capability) when the player connects
              to the new server so that it may re-join its sync-group.
```

Command "stat"

Request a STAT update from the player

Command: "vfdc"

Sends VFD data to the client.

The data is the vfd data to send in the same format as for the SLIMP3 protocol.

Command: "vers"

Sends server version string to the client.

The data is the human readable version information for the server (a simple string)

Command "visu"

Tells the server to activate/deactivate visualizer for the music.

```
$which      1 byte    Which visualizer to use
$count      1 byte    How many parameters there are
$param_1    4 bytes   first parameter...
$param_N    4 bytes   last parameter
```

Values for $which:

```
0    Blank
1    Vumeter
2    Spectrum
3    Waveform
```

```
$channels    0=stereo 1=mono
$style       0=digital 1=analog
$position    left/right position in pixels (0=left side)
$width       width in pixels
$r_position  left/right position of the right channel (stereo only)
$r_width     width in pixels of the right channel (stereo only)2    Spectrum analyzer
$channels   as for vumeter
$bandwidth   0=0..22050Hz, 1=0..11025Hz
$Preemphasis in dB per KHz
$position    left/right position in pixels (0=left side)
$width       width in pixels   $orientation 0=left to right  1=right to left
$bar_width   Bar width in pixels   $bar_space   Bar spacing in pixels
$clipping    0 = show all subbands  1= clip higher bands
$intensity   Bar intensity (greyscale) 1-3
$cap         Bar cap intensity (greyscale) 1-3For stereo, repeat the channel-specific
$r_position  Right side position   ...
$r_cap       Right side bar cap intensity
```

Undocumented commands, brief rundown

Command "audc"

Transporter only, update clock source

Command "audp"

Transporter only, update audio source

Command "body"

Request part of the body of a file from the player (to get remotely downloaded playlists, find bit rates of mp3, etc)

Command "cont"

Content-type related, related to playing remote songs

Command "grfd"

SqueezeboxG only, draw graphics

Command "irtm"

Send timing info to client about when IR messages are processed

Command "knoa"

Knob on a remote control?

Command "knob"

Transporter only, knob-related

Command "rhap"

Rhapsody-specific

Command "rsps"

Transporter only, adjust RS232 baud rate

Command "rstx"

Transporter only, send RS232 TX

Command "setd"

Get/set player firmware settings

Command "upda"

Command "updn"

Command "ureq"

The last 3 are firmware update related commands

Retrieved from "http://wiki.slimdevices.com/index.php?title=SlimProto_TCP_protocol&oldid=13985"
Categories: Development | Networking