SlimProtoDeveloperGuide

From SqueezeboxWiki

Contents

The SlimProto Developer Guide

This page is intended to help interested developers in their quest to understand the SlimProto TCP protocol.

The SlimProto TCP Protocol itself is documented in the help section of your SlimServer install (and copied in a slightly wiki-fied form to SlimProtoTCPProtocol), however the document is a little out of date and lacks certain key pieces of information. With luck this page should go some way to fill ing the gaps.

Rather than list the gaps and continually cross reference, this page will run through a step by step guide to client interaction.

Step 1 - Connect to the server

The first thing a client should do is connect to the TCP port (3483) of the server. It is possible to use the UDP discovery packet as discussed in the SLIMP3ClientProtocol document, but this is not essential.

Step 2 - Introduce yourself

Send a 'HELO' message to establish your identity as a client. The SlimProtoTCPProtocol document is up to date in this respect. Device Ids currently (as of version 7.1) run to 9 and are enumerated as follows.

Id	Device Name
0.	undef
1.	undef
2.	squeezebox
3.	softsqueeze
4.	squeezebox2
5.	transporter
6.	softsqueeze3
7.	receiver
8.	squeezeslave
9.	controller

If your client aims to look appear to be a SlimDevices hardware model such as a SqueezBox2 then be sure to give a current revision or you may find that the server tries to upgrade you (clever beast that SlimServer).

Step 3 - Wait for instructions.

Assuming that you have correctly identified yourself. The SlimServer will start to send you packets. These are mostly graphic packets to show on the display of a device.

Packets you are likely to get include

strm (length <variable> STRM+fixed data+HTTP get)

The STRM packet is the most important message a client can get. It informs the client as to what the server is doing to the stream and (where appropriate) how to retrieve the stream.

Details of the STRM Message format

visu (length 30 - VISU+26 bytes)

Currently undocumented. No information on this is available.

grfe (length 1288 - GRFE+1284 bytes)

Display data. Currently undocumented. The best guess is that this is a bitmap of the 320x32 display (320x32 =10240. divide by 8 (bits in a byte) = 1280) but this has not been tested by myself (nhorlock).

grfb (length 6 - GRFB+2 bytes)

Currently undocumented. Apparently used to set display brightness on SqueezeboxG and Squeezebox2.

audg (length 13 - AUDG+9 bytes)

Currently undocumented. Apparently used to set volume on Squeezebox2.

At the present time I have no information on the Squeezebox2 display format, it appears to be quite different to the earlier models

Step 4 - React to instructions

For the moment I will ignore everything except the STRM message.

As you will see from details of the STRM message format, there are 4 basic forms of the strm message.

S - Start

The start message indicates that the server has started playing and can occur either as a response to a Skip instruction (**STMu**/**STMd** - see STAT) or can be server-initiated as either the initial state of the stream for a reconnecting client or following user intervention (e.g. clicking on the track in the web browser interface).

Actions taken depend upon the current state (and your client to some extent) Current state == Playing If you are already playing, then the present stream should probably be terminated, buffers flushed and a new stream connection made using the given URL. This may vary depending on your clients attitude towards buffering. Current state == Paused If you are paused then you should probably treat this as above, discard your buffers and start afresh. Current state == Stopped Easy one, Connect to the URL and start streaming.

How to connect to the stream. Using the port and IP (if given) specific in the start message, establish a new TCP/HTTP connection. The server is expecting a valid HTTP/1.0 GET string so send the command as given in the strm message.

As a response you will receive an HTTP header followed by the streamed data of the appropriate format (Make sure you are ready for the appropriate stream type and don't let the stream.mp3 URL throw you off the scent, it may not be MP3 at all, always obey the format byte!

Once you have connected to the stream you should send back a response STAT message with command code **STMs** (see STAT for details). This lets the server know that you've started playback

There are other STAT messages that may be applied around this time, **STMh** for example announces that you've processed headers, it doesn't appear to serve any purpose at the time of writing though.

P - Pause

If you are playing then you should simply suspend playback. Retain all the stream context ie. do not flush any buffers and remember you place.

A **STMp** (see STAT) should be sent to indicate that you have paused. This appears to be ignored at present and may not be necessary but you may well break in the future if you take the easy option now.

Ignore format byte etc. It is always mp3 and is effectively unused

U - Unpause

The unpause appears to be overloaded. If you are paused, then clearly this is you instruction to get on with it. Restart playback from the point you stopped and continue with the existing stream. nce playback has restarted, send a STAT/**STMr** to let the server know that you have complied.

If however you are currently playing then the Unpause indicates that the server has finished streaming and you have entered "playout" mode. In playout-mode the server is waiting for the client to tell it that it has completed playback and that it wants the next song.

Bear in mind that the status reflected on the server is controlled by the server side and thus if you now instruct the server to commence the next song but in fact you still have a large buffer of data to play out then the server will show a "Current paying" track that differs from what user is hearing.

Once you are in playout mode expect the stream to end shortly (or to have already ended depending on your read bias) and respond with a STAT/**STMd** (see STAT) message to let the server know when you are ready for more. SlimDevices show **STMd** as decoder ready in their source (This message was added after the last revision of the SlimProtoTCPProtocol document).

Q - Stop/Quit

Do as you are told. Close the stream, flush your buffers, and await further instructions

Step 5 - Process stream and provide feedback.

The GET request made to the steaming socket should be greeted with an HTTP response the body of which will be the data steam itself.

Retrieved from "http://wiki.slimdevices.com/index.php?title=SlimProtoDeveloperGuide&oldid=5477"
Category: Development