0.35

This app (squeeze2upnp - sq2u in short)allow uPNP devices to be used and controlled from LMS. It acts as a proxy between LMS and uPnP/DLNA devices. It scans local network for uPNP/DLNA device and creates an instance of a software player (type squeeezeplay) for each uPNP player found.

It can run on any computer, co-located with LMS or not. For speed reasons, it is better to be co-located, but not mandatory

There are 4 modes of operation:.

STREAM (2) : the main mode

In that mode, sq2u receives all commands and audio streams from LMS, buffers them and then forward them to the uPNP player, without changing the audio coding format.

That buffering process creates two files as large as the original file sent by LMS (be careful with live streams!). By default, these files are created in the directory where sq2u is started from, so write rights are required. Parameter <buffer_dir> can change that location.

All commands sent by LMS are transformed into uPNP commands. Timing (position) information are taken every second from the uPNP player so the LMS cursor might go back and forth a bit as LMS first tries to guess the position and the "match" it with what it receives.

Gapless play is "attempted" by using a uPNP feature that allows sending of data of the "next track/URI" while paying the current one.

Reply Gain is not available, nor Smart Transition

uPNP devices being un-standardized animals, there are various oddities with the translation of the commands, tuned by parameters described in a XML config file described below.

DIRECT(1) : the not-so-compatible-mode (try to avoid using it)

In that mode, sq2u only handles the commands from LMS, not the audio data. Stream data are directly exchanged between LMS and the uPNP player.

This has the benefits of not creating temporary files, but there are a lot of oddities because the content of streams is not seen by sq2u. Typically, playlist are difficult to handle as sq2u has very little information when the move to the next track happens (there is no "URI change" notification, as in mode STREAM). Track change detection is guessed by looking at the time reported by the player and detecting a rollover, but this is very unsatisfactory.

LMS server developers, if you read this, if you'd allow two different URIs to be sent to LMS for the same player, that would solve this problem ☺ (e.g http://<LMS>stream-0.mp3?<mac address> and http://<LMS>stream-1.mp3?<mac address>

Some uPNP devices (e.g. Sonos) are unhappy with the stream being sent directly by LMS, I don't know why at that stage, but they fail in that mode

FULL(3) : for later, if you folks want it

This mode is not implemented, but the idea is that not only sq2u will receive all commands and encoded streams, but it will decode the streams and then re-encoded them into another format (optionally).

This will have the benefit of allowing replay gain, smart transition, resample, and other DSD goodies. Of course, this will be CPU hungry as each uPNP player will create a full decoded/encode process

LMS_UPNP (0) : the not so useful mode, but that was easy

LMS has the possibility to be a uPNP/DLNA server using a plugin developed by Andy Grundman. In that case, if a uPNP device opens http://<LMS>/stream.mp3, LMS will present a virtual player named from mac address of the uPNP device.

This acts as a pseudo-live stream of what LMS plays on that that virtual player. The virtual player is not managed at all by sq2u and cannot be controlled much either by LMS (to my knowledge) - for example, volume cannot be set.

There is a large time (many seconds) gap usually between any LMS command and the result on the player. This is probably due to buffering of the pseudo-live stream by the player.

This LMS feature is normally used by manually adding (e.g.) a radio stream on your uPNP player device that points to http://<LMS>/stream.mp3.

In that mode, sq2u does not do much except trying to automatically set the URI on the uPNP player and request a "play". It does not see any LMS commands nor data stream.

This is a simple helper to avoid doing that manually on players. Some uPNP devices (e.g. Sonos) do not like that and want the LMS URI to be as a "radio". The problem with Sonos is probably similar to what happens in DIRECT mode and is maybe linked to the size of the stream sent by LMS

Comments & assumptions

- A searches for new uPNP devices will occur every 2 minutes and last for 15s. 1st search will last 15s before slimdevices are created – be patient ☺
- If a player disappears from the LAN, it will be automatically removed after 2 minutes maximum.
- There a few hidden commands that can be entered WHILE sq2u is RUNNING (this is NOT command line options)
    - exit : terminates
    - save <filename> : save the config file
    - stop * or stop <string> will stop any player whose name matches <string>

- Use the "–h" option to know startup option on the command line
    - –s <server:port> : force LMS server & port
    - –x <config file> : set config from <config file>. Default is ./config.xml

    - –f <logfile> : set log to be in <logfile> instead of stderr
    - –d : set various logvele (see –h for detail)
    - –z : run daemonized (Linux only …)
- Command Line Interface (CLI) is used and assumes port 9090, no login/password at that time. This is only needed for the <seek_after_pause> option to work (see below), nothing else
- If the volume is changed directly on the uPNP player, it will NOT be reflected in LMS, not will be a pause/stop action made directly on the player – trying to command the player from two sides is problematic at this stage
- Switching off the player in LMS will disable management, so such palyer can then be controlled by any other application but still is enable in LMS
- Tests have been made with mp3, flac and pcm, success with other format is unlikely vary, so it recommended to let LMS do the translation beyond these 3 codecs.
- With a Sonos system, it works very well and it even works when 2 speakers are paired together – just disable one of them from sq2u (the one that does not work anyway - the "slave" of the pair)
- With other upnp/dlna, mileage varies. I have a Marantz NR1603 and it now works well including pause, cue, gapless, flac, mp3, pcm …
- If you use 24 or 32 sample size, some players might not support it so check the logs and make sure you enable transcoding in LMS. For example, Sonos does not support 24 bits flac. Look at the packing type of 24 bits format supported by your player as weel (<L24_format>
- There is a Windows and Linux x86 version – they can run on any computer, it does not need to be the LMS server. There is no ARM version
- No test have been made with LMS with a login/password

Configuration

Upon startup, sq2u tries to open a file named "config.xml" in the current directory (or uses the filename provided using the –x option). If no file is found, all parameters will be set to default. To generate such a config file, simply type "save <filename>" when sq2u is running and a file with all existing configuration parameters and discovered devices will be created.

!!!! Do not try to create your 1st config file manually and guess default parameters, launch sq2u once, wait 30 seconds and then type "save <filename> and modify the generated file !!!!

XML config file options

<?xml version="1.0"?>
<squeeze2upnp>

Global options
<server/>[ip:port | ?]</server>
The name:port of your LMS server – remove the option or put a ? for auto-discover

<slimproto_stream_port>[port]</slimproto_stream_port>
Only used for the LMS_UPNP mode to specify the port. 9000 is default
Mode : 0

<base_mac>[xx:xx:xx:xx:xx:xx]</base_mac>
By default, the MAC of the uPNP player is used to identify it. In case sq2u cannot read this MAC address, then the first discover player will start with that <base_mac>, the 2nd player with <base_mac> + 1 and so on.

<slimproto_log>[error | warn | info | debug | sdebug]</slimproto_log>
<stream_log>[error | warn | info | debug | sdebug]</stream_log>
<output_log>[error | warn | info | debug | sdebug]</output_log>
<decode_log>[error | warn | info | debug | sdebug]</decode_log>
<web_log>[error | warn | info | debug | sdebug]</web_log>
<upnp_log>[error | warn | info | debug | sdebug]</upnp_log>
<main_log>[error | warn | info | debug | sdebug]</main_log>
<sq2mr_log>[error | warn | info | debug | sdebug]</sq2mr_log>
Level of debugging for each sub-item of sq2upnp

<common>
All items in that sub-section set the default behavior for all managed players. Any option can be repeated in the <device> section of a given detected player to overload the <common> options

<enabled>[0 | 1]</enabled>
'1' to automatically add any discovered uPNP player to LMS players, even if <device> section referring to it

<streambuf_size>[value > 1 000 000]</streambuf_size>
Size of the internal memory buffer used to store audio data sent by LMS. Any value above 1 000 000 is fine, 2 000 000 is recommended. Not useful to change unless you experience a lot of issues with slow networks
Modes : 2, 3

<output_size>[value > 2 000 000]</output_size>
Size of the internal memory to re-encoding.
Modes: 3

<buffer_dir>[path with /]</output_size>
Directory where the stream data are buffered. Two files are created for each player : xx-xx-xx-xx-xx-xx-idx-0 and xx-xx-xx-xx-xx-xx-idx-1 where xx-xx-xx-xx-xx-xx if the MAC address of the player. These files are as big as the largest file played by LMS. In case of live stream, these file grow constantly, so be careful. Just press "next" on the live stream from time to time will solve the problem (or see below)
Modes: 2, 3

<buffer_size>[-1 | value]</buffer_size>
Maximum size of each stream data buffer. When a buffer reaches the set size, it is shrunk to half. As a consequence, sq2upnp "loses" memory of every received data before the shrinkage happens and cannot seek back to any of "lost" portion. In some rare cases, that might prevent a "reposition" or a "pause" to work properly. Note that the shrinkage only happens if more than half of the buffered data has already been passed to the player. If a player is "paused" on

a live stream, the buffer will continue to grow until the player is “un-paused”. I might add, if needed, an option to force the shrinkage to happen. -1 let the buffer grow without a limit.
Modes: 2, 3
(not implemented before 0.1)

<stream_length>[-1 | -3 | -4 | value > 100 000 000]</stream_length>
-1 : infinite transfer
-3 : chunked transfers ➔ try that 1st
-4 : transfer up to close
value : any large number
How to do HTTP transfers. LMS does not tell in advance the length (in either bytes or in seconds) of the audio stream that it will send. Normally, HTTP transfer should use -3 which allow data to be transferred by chunks until there is nothing to transfer. When there is nothing left, sq2u sends an empty chunk so that of the stream shall be assumed. But some players do not seem to like “chunked” mode so just put there a value larger than the largest of your audio encoded tracks should be set here - try between 100 000 000 and 1 000 000 000. Typically, Sonos does not accept chunked HTTP transfer with MP3 files. Don’t know why, but I’m not the first one to face that.
Modes: 2, 3

<max_read_wait>[value > 10]</max_read_wait>
When the buffer of data from LMS is empty but more data are expected from LMS, this option sets the amount of time in Nx50ms that sq2u waits before signaling end of stream to the player (which would be an overrun)
Modes: 2, 3

<max_GET_bytes>[0 | -1 | value > 100 000]</max_GET_bytes>
Limit the size of chunk of stream data sent by sq2u to a player. This helps slowing down internal sq2u buffer consumption and preventing overrun, especially with slow networks.
0 = LMS “recommended” value
-1 = Max sq2u can send (1MB)
Modes: 2, 3

<didle_duration>24:00:00.000</didle_duration>
(N/A)

<process_mode>[0 … 3]</process_mode>
The mode of operation
0=LMS_UPNP
1=DIRECT
2=STREAM ➔ use only this one for now
3=FULL

<can_pause>[0 | 1]</can_pause>
Some players cannot properly pause. Set to 0 will ignore LMS pause requests. The most useful parameter to try to emulate pause is <seek_after_pause>
Modes 1, 2, 3

<seek_after_pause>[0 | 1]</seek_after_pause>

When they “unpause”, players request the same data stream again and server always restarts from the beginning (this is normal HTTP-stream mode). Normal players will (e.g. Sonos) either consume the data till they reach again the “paused point” or do GET with the right offset. Some players do not request properly the offset and the paused track restart from the beginning. Set to 1, sq2upnp handles this offset on behalf of the player. This uses the CLI interface to instruct LMS to cue to the paused position, so if you have a login/password set, it will fail as there is no option today to have sq2u to use login/password
Modes 1, 2, 3

<force_volume>[0 | 1]</force_volume>
Some players, do not accept the ramp-up/down volume commands sent by LMS at each stop/pause/start and will miss some of these volume commands. Set to 1 this parameters forces sq2u to re-issue a “set volume” command after the player has confirmed that a track has started to play
Modes: 1, 2, 3

<volume_on_play>[0 | 1]</volume_on_play>
LMS sends a volume control command (set to 0) whenever a new player is registered (at startup or anytime). This can be an annoying and unwanted effect when such player is not actively controlled by LMS (although it is registered). One way to avoid that is to set the player “off” in LMS, but ser might want to change player control without having to switch it “off” in LMS. With that parameter to 1, sq2u will ignore any volume command from LMS until it is requested to “play” a track (volume command are still memorized, though). Note also that when “off” or when stopped, sq2u will ignore all uPnP event to avoid unsolicited reaction from LMS when a uPNP player is being controller by another mean than LMS.
Modes: 1, 2, 3

<volume_curve> [xx:yy, xx:yy, …] </volume_curve>
LMS sends volume values from 0 to 65536, although on the user interface, it says from 0 to 100%. The translation between the two values is log, not linear and is player dependant (i.e. 20% is 700, 50% = 3800 and 80% is 21000). This parameter re-linearizes the translation for a squeezeplay device. This is a comma-separated list of pair [send_value:pourcentage]. Any LMS request between two points is a linear approximation between these. Change it to match your taste of volume command
Modes: 1, 2, 3

<accept_nexturi>[0 | 1]</ accept_nexturi >
Gapless play is done by using the possibility to set a “current track/URI” and a “next track/URI” in uPNP players. When they reach the end of “current track/URI”, players move to the next one automatically. Some players (very few, more software renderers probably) do not support this feature and will stop when the “current track/URI” ends. This options forces the next track to be played on faulty players, but there will be a gap
NB : good players (e.g. Sonos) buffer the “next track/URI” while playing the “currentURI” to do real gapless. Average players, although they support “next track/URI” wait for the end of the current track before buffering the next one, and that will create a gap.
Modes: 2, 3

<codecs>[codec1, codec2, …]</codecs>

The comma separated list of codecs, supported by the uPNP player, that will be reported to LMS. If a track is encoded in a non-supported format, LMS will apply its transcoding rules. Recommended is mp3 and flc and pcm (i.e: flc,mp3, pcm) currently. Other choices are: wma, wmapp, wmal, aif, alc, aac but they have not been tested at all.
When LMS cues, it generates locally a new "truncated" file and sends a new command to play/stream that file. Unfortunately, it does not seem to insert a header on the truncated file. Some uPNP players do not like that, so a header is "guessed" and re-inserted by sq2u. This is only done for flac, so other format might be problematic.
PCM byte ordering (endianess) is different between LMS and all players I've tried – sq2u reverse by ordering for 16, 24 and 32 bits samples when it is sending a PCM file.
Modes: 2, 3

<sample_rate>[32000 | 44100 | 48000 | 96000]</sample_rate>
Maximum sample rate, accepted by the uPNP player, that will be reported to LMS. If a track is encoded at a higher rate, LMS will apply resampling rules.
Modes: 1, 2, 3

<L24_format>[0 | 1 | 2 |3]</L24_format>
Defines what kind of 24 bits sample the player can handle.
0: packed format with 3 bytes only
1: packed format according to LPCM type 1 packing
2: unpack 3 bytes on 4 bytes, fill lower byte with 0 (not supported yet)
3: unpack 3 bytes on 4 bytes, fill higher byte with 0 (not supported yet)
Modes: 1, 2, 3

</common>

This is the beginning of the sections that describes each device. There will be one <device> section per each uPNP player. Any parameter set in the <common> can be repeated here, to overload the defaults.

first device …

<device>

<udn>uuid:RINCON_000E58FD8A4A01400</udn>
!!! do not edit !!!
The uPNP device unique identifier for that player – this is discovered automatically. Here is just an example of a Sonos player

<name>[string]</name>
The name of the player as seen in LMS – can be edited to what you want. It is the "uPnP friendly" name by default
Modes: 1, 2, 3

<enabled>[0 | 1]</enabled>
Set to 1 to allow the uPNP player to be managed be sq2upnp. A LMS device will be created for each player whose mode is above 0. Set to 0 to ignore the player
Modes: 0, 1, 2, 3

```
</device>

next device …
<device>
…
</device>

</squeeze2upnp>
```

Config example

```
<?xml version="1.0"?>

<squeeze2upnp>

<server>?</server>
<slimproto_stream_port>9000</slimproto_stream_port>
<base_mac>00:01:02:03:04:05</base_mac>
<add_unknown>1</add_unknown>
<slimproto_log>info</slimproto_log>
<stream_log>info</stream_log>
<output_log>info</output_log>
<decode_log>info</decode_log>
<web_log>info</web_log>
<upnp_log>info</upnp_log>
<main_log>info</main_log>
<sq2mr_log>info</sq2mr_log>

<common>
        <streambuf_size>2097152</streambuf_size>
        <output_size>3528000</output_size>
        <buffer_dir>.</buffer_dir>
        <stream_length>-3</stream_length>
        <max_read_wait>100</max_read_wait>
        <max_GET_bytes>1</max_GET_bytes>
        <didle_duration>24:00:00.000</didle_duration>
        <process_mode>2</process_mode>
        <can_pause>1</can_pause>
        <codecs>mp3</codecs>
        <sample_rate>48000</sample_rate>
        <seek_after_pause>0</seek_after_pause>
        <force_volume>1</force_volume>
        <volume_curve>0:0, 400:10, 700:20, 1200:30, 2050:40, 3800:50, 6600:60, 12000:70,
21000:80, 37000:90,65535:100</volume_curve>
        <accept_nexturi>1</accept_nexturi>
</common>

<device>
        <udn>uuid:RINCON_000E58C202B601400</udn>
```

```
        <name>192.168.2.24 – Sonos PLAY:1</name>
        <enabled>0</enabled>
</device>

<device>
        <udn>uuid:RINCON_000E58FD8A4A01400</udn>
        <name>192.168.2.23 - Sonos PLAY:3</name>
        <enabled>1</enabled>
        <codecs>flc,mp3</codecs>
</device>

<device>
        <udn>uuid:RINCON_000E58C3F67C01400</udn>
        <name>192.168.2.22 - Sonos PLAY:1</name>
        <enabled>1</enabled>
        <codecs>flc,mp3</codecs>
</device>

<device>
        <udn>uuid:5f9ec1b3-ff59-19bb-8530-000678110596</udn>
        <name>marantz NR1603</name>
        <enabled>1</enabled>
        <codecs>flc,mp3,pcm</codecs>
</device>

<device>
        <udn>uuid:5f9ec1b3-ed59-1900-4530-9c645e0296f5</udn>
        <name>JBL OnBeat Air0296F5</name>
        <enabled>1</enabled>
        <codecs>mp3,pcm</codecs>
</device>

</squeeze2upnp>
```