NAME

jack_audio_receive − JACK client to transmit audio data in a LAN

SYNOPSIS

jack_audio_receive [OPTIONS] listening_port

DESCRIPTION

jack_audio_send & *jack_audio_receive* are JACK clients allowing to transmit uncompressed native JACK 32 bit float audio data on the network using UDP OSC messages.

All involved JACK servers must share the same sampling rate but can run at different period sizes. NO resampling is involved. Diverging clocks of distributed audio interfaces can pose an issue.

- The main purpose is to transmit audio in one direction from host a to host(s) b(, c, ...)
- Sender and receiver are seperate programs
- There is no such thing as master / slave
- The programs run as a regular JACK clients
- There is no resampling done at either the sender or receiver side

The term "mc" period refers to multichannel period which contains the period of each channel for a given JACK cycle.

audio_rxtx uses liblo >= 0.28 with a maximum message size (UDP) of 65k

THIS PROGRAM COMES WITHOUT ANY WARRANTY

jack_audio_receive version is 0.82

OPTIONS

−−**help** (w/o argument)
Display help and quit.

−−**version** (w/o argument)
Display version and quit.

−−**loinfo** (w/o argument)
Display liblo props and quit.

−−**out** (integer)
Number of output/playback channels. Default: 2

−−**offset** (integer)
Ignore first n (=offset) channels from incoming audio data. This allows a receiver to pick a specific (contiguous) channel range from the input. Default: 0

−−**connect** (w/o argument)
Autoconnect ports. This will connect all jack_audio_receive ports to physical output/playback ports. Default: off

−−**16** (w/o argument)
convert JACK 32 bit float wave data to 16 bit integer (˜PCM) data for the network transmission. This can save some bandwidth and/or allow more channels with large period sizes. Default: 32 bit float

−−**name** (string)
Jack client name. Default: jack_audio_receive

−−**sname** (string)
JACK server name to start jack_audio_receive in a specific JACK server. The env variable $JACK_DEFAULT_SEVER is supported. Default: default

−−**pre** (integer)

Initial buffer size. Default: 4 mc periods

−−**max** (integer)
Max buffer size (>= init). Default: (auto)

−−**rere** (w/o argument)
Rebuffer on sender restart. Default: off

−−**reuf** (w/o argument)
Rebuffer on underflow. Default: off

−−**nozero** (w/o argument)
Re−use old data on underflow. Default: off

−−**norbc** (w/o argument)
Disallow external buffer control via /buffer ii Default: off (allow)

−−−**update** (integer)
Update info displayed in terminal every nth JACK cycle. Default: 99

−−**limit** (integer)
Limit totally received messages. This is useful for tests. Receive an exact amount of messages, then quit program. Default: off

−−**quiet** (w/o argument)
Don’t display running info Default: off

−−**shutup** (w/o argument)
Don’t output anything on std{out,err} except fatal errors. Default: off

−−**io** (w/o argument)
Enable Remote Control / GUI. Default: off

−−**nopush** (w/o argument)
Disable push to GUI. Default: off

−−**iohost** (string)
GUI host. Default: localhost

−−**ioport** (string)
GUI port (UDP). Default: 20220

−−**close** (w/o argument)
Quit if received data is incompatible. Default: off

listening_port (integer)
Local port to listen for audio.

EXAMPLES

Receive 2 channels on port 1234, automatically connect ports, rebuffer on underflow:

$ jack_audio_receive −−out 4 −−connect −−rere 4444

Example output of jack_audio_receive:

receiving on UDP port: 4444
started JACK client 'receive' on server 'default'
sample rate: 44100
bytes per sample: 4
period size: 128 samples (2.902 ms, 512 bytes)
channels (playback): 4
channel offset: 0
multi−channel period size: 2048 bytes
underflow strategy: fill with zero (silence)

rebuffer on sender restart: yes
rebuffer on underflow: no
allow external buffer control: yes
shutdown receiver when incompatible data received: no
initial buffer size: 4 mc periods (11.610 ms, 8192 bytes, 0.01 mb)
allocated buffer size: 177 mc periods (513.742 ms, 362496 bytes, 0.36 mb)

autoconnect: receive:output_1 −> firewire_pcm:000a9200d6012385_MainOut 1L_out
autoconnect: receive:output_2 −> firewire_pcm:000a9200d6012385_MainOut 2R_out
autoconnect: receive:output_3 −> firewire_pcm:000a9200d6012385_LineOut 3L_out
autoconnect: receive:output_4 −> firewire_pcm:000a9200d6012385_LineOut 4R_out

sender was (re)started. equal sender and receiver period size

5048 i: 4 f: 4.2 b: 8704 s: 0.0123 i: 2.90 r: 0 l: 0 d: 0 o: 0 p: 0.0

Legend:

- #: message id given by sender, sequence number since start of sender
- i: input channel count (can dynamically change)
- f: buffer fill level: periods (for all channels)
- b: buffer fill level: bytes
- s: buffer fill level: seconds
- i: average time between messages: milliseconds
- r: remote xrun counter
- l: local xrun counter
- d: dropped multi−channel periods (buffer underflow)
- o: buffer overflows (lost audio)
- p: how much of the available process cycle time was used to do the work (1=100%)

Receive max 8 channels ignoring the first 2 incoming channels, as 16 bit data, on port 1234, pre−buffer 100 mc periods before playing, allow max 2000 mc periods buffer, close receiver when incompatible:

$ jack_audio_receive −−out 8 −−offset 2 −−16 −−pre 100 −−max 2000 −−close 1234

ERROR MESSAGES

jack_audio_receive does not automatically start a JACK default server if there is none running. This will lead to the following message:

Cannot connect to server socket err = No such file or directory Cannot connect to server request channel jack server is not running or cannot be started jack_client_open() failed, status = 0x11 Unable to connect to JACK server

Simply start JACK before using jack_audio_receive

PROGRAM STATUSES

jack_audio_receive statuses:

0) initializing, starting up with given parameters

1) waiting for audio (if no sender is currently active)

2 − 4 only if sender was started **without** −−nopause:

2) receiving audio **/offer** from sender

3) **/deny** transmission (if offered audio was incompatible)

−> don't send /deny to sender if receiver was started with −−close
−> quit receiver when started with −−close

OR

4) **/accept** transmission (if offered audio was compatible)

5) buffering audio (for the given −−pre size in periods)

6) playing audio (read from buffer, pass to jack)

7) buffer underflow (not enough data to read)

−> rebuffer (if −−reuf set)
−> fill buffer with zero (silence) (if −−nozero NOT set)
−> fill buffer with last available value (if −−nozero set)

8) buffer overflow (buffer full, can’t add more data)

9) sender was restarted

−> rebuffer (if −−rere set)

10) incompatible data received

−> telling sender to stop (if −−close NOT set)
−> shutting down receiver (if −−close set)

11) receiver closed (ctrl+c / −−limit reached)

−> telling sender to pause (if −−close NOT set)

OSC FORMAT VERSION 1.0

The OSC messages that are sent by jack_audio_received are defined as follows:

/accept

(no parameters)

/deny fi

1) f: format version
2) i: sampling rate

/pause

(no parameters)

All properties refer to the receiving host.

The OSC messages that are understood by jack_audio_receive are defined as follows:

- **/offer ffiiifh**
- **/audio hhtib***
- **/buffer ii**

  1) i: buffer pre−fill (−−pre)
  2) i: buffer max size (−−max)

/buffer is not sent by sender, it’s an experimental way for any process to control the buffer status of the receiver via OSC.

Playback will either pause (refill buffer) or audio will get lost (drop samples) to match the desired buffer fill level.

If <buffer max size> is not the same as −−max / auto, a new buffer will be created and filled with <buffer pre−fill>

Please also see manpage of jack_audio_send. The liblo tool programs *oscdump* and *oscsend* should also be mentioned here.

RESOURCES

Github: https://github.com/7890/jack_tools

BUGS

Please report any bugs as issues to the github repository. Patches and pull requests are welcome.

SEE ALSO

jack_audio_receive(1) **jackd**(1) **jack_netsource**(1) **jacktrip**(1) **zita−njbridge(1)**

AUTHORS

Thomas Brand <tom@trellis.ch>

COPYING

Copyright (C) 2013 − 2014 Thomas Brand. Free use of this software is granted under the terms of the GNU General Public License (GPL).