LinuxSampler Developer's
Internet Draft
Document: draft-linuxsampler-protocol-05.txt
Expires: June 2004

C. Schoenebeck
<Affiliation>

Wednesday, May 19, 2004

LinuxSampler Control Protocol

Status of this Memo

This document specifies an application specific protocol for the LinuxSampler core application and arbitrary third party software that interacts with the LinuxSampler application, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited. THIS DOCUMENT IS ONLY AN INITIAL DRAFT NOT A FINAL VERSION OF THE PROTOCOL!

Abstract

The LinuxSampler Control Protocol (LSCP) is an application-level protocol primarily intended for local and remote controlling the LinuxSampler main application, which is a sophisticated console application essentially playing back audio samples and manipulating the samples in real time to certain extent.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [1].

This protocol is always case-sensitive if not explicitly claimed the opposite.

In examples, "C:" and "S:" indicate lines sent by the client (frontend) and server (LinuxSampler) respectively. Lines in examples must be interpreted as every line being CRLF terminated (carriage return character followed by line feed character as defined in the ASCII standard), thus the following example:

```
C: “some line”
   “another line”
```

must actually be interpreted as client sending the following message:

“some line<CR><LF>another line<CR><LF>”

where <CR> symbolizes the carriage return character and <LF> the line feed character as defined in the ASCII standard.

Due to technical reasons, messages can arbitrary be fragmented, means the following example:

S: “abcd”

could also happen to be sent in three messages like in the following sequence scenario:

- server sending message “a”
- followed by a delay (pause) with arbitrary duration
- followed by server sending messsage “bcd<CR>”
- again followed by a delay (pause) with arbitrary duration
- followed by server sending the message “<LF>”

where again <CR> and <LF> symbolize the carriage return and line feed characters respectively.

Table of Contents

1. Introduction

LinuxSampler is a so called software sampler application capable to playback audio samples from a computer's Random Acess Memory (RAM) as well as directly streaming it from disk. LinuxSampler is designed to be modular. It provides several so called “sampler engines” where each engine is specialized for a certain purpose. LinuxSampler has virtual channels which will be referred in this document as “sampler channels”. The channels are in such way virtual as they can be connected to an arbitrary MIDI input method and arbitrary MIDI channel (e.g. sampler channel 17 could be connected to an ALSA sequencer device 64:0 and listening to MIDI channel 1 there). Each sampler engine will be assigned an own instantance of one of the available sampler engines (e.g. GigEngine, DLSEngine). The audio output of each sampler channel can be routed to an arbitrary audio output method (ALSA / JACK) and an arbitrary audio output channel there.

2. Communication Overview

There are two distinct methods of communication between a running instance of LinuxSampler and one or more control applications, so called “frontends”: a simple TCP unidirectional communication method and a TCP / UDP combination for bidirectional communication. The latter needs more effort to be implemented in the frontend application. The two communication methods will be described next.

2.1 Simple unidirectional communication

This simple communication method is primarily based on TCP. The frontend application establishes a TCP connection to the LinuxSampler instance on a certain host system. Then the frontend application will send certain ASCII based commands as defined in this document (every command line must be CRLF terminated – see “Conventions used in this document” at the beginning of this document) and the LinuxSampler application will response after a certain process time with an appropriate ASCII based answer, also as defined in this document. So this TCP communication is simply based on query and answer paradigm. That way LinuxSampler is only able to answer on queries from frontends, but not able to automatically send messages to the client if it's not asked to. The fronted should not reconnect to LinuxSampler for every single commmand, instead it should keep the connection established and simply resend message(s) for subsequent commands. To keep LinuxSampler's informations in the frontend up-to-date the frontend has to periodically send update commands to get the current informations of the LinuxSampler instance. This is often referred as “polling”. The disadvantage of this simple unidirectional communication approach is obvious: it means network traffic overhead and introduces latency regarding the update of the informations, but is very simple to implement.

2.2 Advanced bidirectional communication

This more sophisticated communication method is actually only an extension of the simple unidirectional communication method. The frontend still uses a TCP connection and sends the same commands on the TCP connection, but the frontend has to provide an open UDP port for receiving event messages from the LinuxSampler application. The frontend has to register it's UDP port to the LinuxSampler application by sending the following command on it's TCP connection:

```
SUBSCRIBE NOTIFICATION <udp-port>
```

where <udp-port> will be replaced by the respective UDP port number. If this is accepted by the LinuxSampler application, the frontend will receive events from that point whenever some for the frontend noteworthy event occurred in the LinuxSampler instance. These event UDP packets usually only contain basic informations like the event category and for example on which sampler channel the event occurred. After receiving the event, the frontend might have to react by issueing a respective update command on it's TCP connection to get the detailed change. This is dependant to the event type and due to the fact that UDP packets are limited to certain packet size (usually < 64 kB). So again, some events provide already an exact information about the new state and some need to be ordered on the

primary TCP connection by the frontend.

Example: the fill states of disk stream buffers have changed on sampler channel 4 and the LinuxSampler instance will react by sending the following UDP packet:

```
CHANGE CHANNEL BUFFER_FILL 4
```

LinuxSampler will not insert the fill states of the buffers into the UDP packet, instead the frontend is forced to acquire this information by sending the following update command:

```
GET CHANNEL BUFFER_FILL PERCENTAGE 4
```

to get the fill states of all disk stream buffers on sampler channel 4 and will receive the following answer from LinuxSampler:

```
"[35]62%,[33]80%,[37]98%"
```

Which means there are currently three active streams on sampler channel 4, where the stream with ID "35" is filled by 62%, stream with ID 33 is filled by 80% and stream with ID 37 is filled by 98%.

Beside normal event packets, LinuxSampler will also periodically send PING packets to check if a frontend is still alive. The frontend has to answer with a PONG UDP package (PING and PONG UDP packages will be defined later in this document). If LinuxSampler will not receive such a PONG packet it will consider the frontend to be not available and remove it from the notification list. Such a PING packet is also sent by LinuxSampler when the frontend issued a "SUBSCRIBE NOTIFICATION" command to check if the given UDP port is really available and not constrained by a firewall for example, so the frontend has to open the input UDP port before it tries to register for notification by sending the mentioned command.

3. Description for control commands

This chapter will describe the available control commands that can be sent on the TCP connection in detail. Some certain commands (e.g. "GET CHANNEL INFO" or "GET ENGINE INFO") lead to multiple-line responses. In this case LinuxSampler signals the end of the response by a "." (single dot) line.

3.1 Configuring audio drivers

Drivers in LinuxSampler are called devices. You can use multiple devices simultaniously, e.g. to output the sound of one sampler

channel using the Alsa audio output driver, and on another sampler channel you might want to use the Jack audio output driver. Usually these devices will be created automatically by LinuxSampler when you select an audio output type on a sampler channel and the respective device was not created yet, but this is not always possible, because some drivers might require explicit parameters (e.g. host name for some audio over ethernet driver) and even if not, LinuxSampler will just use default settings when it has to automatically create a device. So the following commands are used to configure LinuxSampler's audio output drivers and their parameters.

Instead of defining commands and parameters for each driver individually, all possible parameters, their meanings and possible values have to be obtained at runtime. This makes the protocol a bit abstract, but has the advantage, that frontends can be written independently of what drivers are implemented and what parameters these drivers are actually offering.

3.1.1 Getting all available audio output drivers

Use the following command to list all audio output drivers currently available for the LinuxSampler instance:

GET AVAILABLE_AUDIO_OUTPUT_TYPES

Possible Answers:

LinuxSampler will answer by sending comma separated character strings symbolizing the available audio output drivers.

Example:

C: “GET AVAILABLE_AUDIO_OUTPUT_TYPES”
S: “Alsa,Jack”

3.1.1 Getting informations about a specific audio output driver

Use the following command to get detailed informations about a specific audio output driver:

GET AUDIO_OUTPUT_TYPE INFO <audio-output-type>

Where <audio-output-type> is the name of the audio output driver, returned by the “GET AVAILABLE_AUDIO_OUTPUT_TYPES” command.

Possible Answers:

LinuxSampler will answer by sending a <CRLF> separated list. Each answer line begins with the information category name followed by a colon and then a space character <SP> and finally the info character string to that info category. At the moment the following information categories are defined:

DESCRIPTION –
character string describing the audio output driver

VERSION -
character string reflecting the driver's version

PARAMETERS –
comma separated list of all parameters available for the given audio output driver, at least parameters 'CHANNELS', 'SAMPLERATE' and 'ACTIVE' are offered by all audio output drivers

The mentioned fields above don't have to be in particular order.

Example:

C: “GET AUDIO_OUTPUT_TYPE INFO Alsa”
S: “DESCRIPTION: Advanced Linux Sound Architecture”
“VERSION: 1.0”
“PARAMETERS: CHANNELS,SAMPLERATE,ACTIVE,FRAGMENTS,FRAGMENTSIZE,CARD”
“.”

3.1.1 Getting informations about specific audio output driver parameter

Use the following command to get detailed informations about a specific audio output driver parameter:

GET AUDIO_OUTPUT_TYPE_PARAMETER INFO <audio-t> <prm> [<deplist>]

Where <audio-t> is the name of the audio output driver as returned by the “GET AVAILABLE_AUDIO_OUTPUT_TYPES” command, <prm> a specific parameter name for which information should be obtained (as returned by the “GET AUDIO_OUTPUT_TYPE INFO” command) and <deplist> is an optional list of parameters on which the sought parameter <prm> depends on, <deplist> is a list of key-value pairs in form of “key1=val1 key2=val2 ...”, where character string values are encapsulated into apostrophes ('). Arguments given with <deplist> which are not dependency parameters of <prm> will be ignored, means the frontend application can simply put all parameters into <deplist> with the values selected by the user.

Possible Answers:

LinuxSampler will answer by sending a <CRLF> separated list. Each answer line begins with the information category name followed by a colon and then a space character <SP> and finally the info character string to that info category. There are informations which are always returned, independently of the given driver parameter and there are optional informations which are only shown dependently to given driver parameter. At the moment the following information categories are defined:

DESCRIPTION –
arbitrary text describing the purpose of the parameter (always returned, no matter which driver parameter)

MANDATORY -
either true or false, defines if this parameter must be given when the device is to be created with the 'CREATE AUDIO_OUTPUT_DEVICE' command (always returned, no matter which driver parameter)

FIX -
either true or false, if false then this parameter can be changed at any time, once the device is created by the 'CREATE AUDIO_OUTPUT_DEVICE' command (always returned, no matter which driver parameter)

MULTIPLICITY -
either true or false, defines if this parameter allows only one value or a list of values, where true means multiple values and false only a single value allowed (always returned, no matter which driver parameter)

DEPENDS -
comma separated list of paramters this parameter depends on, means the values for fields 'DEFAULT', 'RANGE_MIN', 'RANGE_MAX' and 'POSSIBILITIES' might depend on these listed parameters, for example assuming that an audio driver (like the Alsa driver) offers parameters 'CARD' and 'SAMPLERATE' then parameter 'SAMPLERATE' would depend on 'CARD' because the possible values for 'SAMPLERATE' depends on the sound card which can be chosen by the 'CARD' parameter (optionally returned, dependent to driver parameter)

DEFAULT –
reflects the default value for this parameter which is used when the device is created and not explicitly given with the 'CREATE AUDIO_OUTPUT_DEVICE' command,

in case of MULTIPLCITY=true, this is a comma separated list, that's why character strings are encapsulated into apostrophes (')
(optionally returned, dependent to driver parameter)

RANGE_MIN –
defines lower limit of the allowed value range for this parameter, can be an integer value as well as a dotted number
(optionally returned, dependent to driver parameter, but always in conjunction with RANGE_MAX)

RANGE_MAX –
defines upper limit of the allowed value range for this parameter, can be an integer value as well as a dotted number
(optionally returned, dependent to driver parameter, but always in conjunction with RANGE_MIN)

POSSIBILITES –
comma separated list of possible values for this parameter, character strings are encapsulated into apostrophes
(optionally returned, dependent to driver parameter)

The mentioned fields above don't have to be in particular order.

Examples:

C: “GET AUDIO_OUTPUT_TYPE_PARAMETER INFO Alsa CARD”
S: “DESCRIPTION: sound card to be used”
“MANDATORY: false”
“FIX: true”
“MULTIPLICITY: false”
“DEFAULT: '0,0'”
“POSSIBILITES: '0,0','1,0','2,0'”
“.”

C: “GET AUDIO_OUTPUT_TYPE_PARAMETER INFO Alsa SAMPLERATE”
S: “DESCRIPTION: output sample rate in Hz”
“MANDATORY: false”
“FIX: false”
“MULTIPLICITY: false”
“DEPENDS: CARD”
“DEFAULT: 44100”
“.”

C: “GET AUDIO_OUTPUT_TYPE_PARAMETER INFO Alsa SAMPLERATE CARD='0,0'”
S: “DESCRIPTION: output sample rate in Hz”

“MANDATORY: false”
“FIX: false”
“MULTIPLICITY: false”
“DEPENDS: CARD”
“DEFAULT: 44100”
“RANGE_MIN: 22050”
“RANGE_MAX: 96000”
“.”

3.1.1 Loading an audio output driver

Use the following command to create a new audio output device for the desired audio output system:

CREATE AUDIO_OUTPUT_DEVICE <audio-output-type> [<param-list>]

Where <audio-output-type> should be replaced by the desired audio output system and <param-list> by an optional list of driver specific parameters in form of “key1=val1 key2=val2 ...”, where character string values should be encapsulated into apostrophes ('). Note that there might be drivers which require parameter(s) to be given with this command. Use the previously described commands in this chapter to get those informations.

Possible Answers:

“OK” -
in case the driver was successfully loaded

“WRN:<warningcode>:<warningmessage>” -
in case the driver was loaded successfully, but there are noteworthy issue(s) related (e.g. sound card doesn't suport given hardware parameters and the driver is using fallback values), providing an appropriate warning code and warning message

“ERR:<errorcode>:<errormessage>” -
in case it failed, providing an appropriate error code and error message

Examples:

C: “CREATE AUDIO_OUTPUT_DEVICE Alsa CARD='1,0' SAMPLERATE=96000”
S: “OK”

C: “CREATE AUDIO_OUTPUT_DEVICE Alsa”
S: “OK”

3.1.1 Unloading an audio output driver

Use the following command to destroy a created output device:

DESTROY AUDIO_OUTPUT_DEVICE <audio-output-type>

Where <audio-output-type> should be replaced by the audio output system name given by the “GET AVAILABLE_AUDIO_OUTPUT_TYPES” command.

Possible Answers:

“OK” -
in case the driver was successfully unloaded

“WRN:<warningcode>:<warningmessage>” -
in case the driver was loaded successfully, but there are noteworthy issue(s) related (e.g. an audio over ethernet driver was unloaded but the other host might not be informed about this situation), providing an appropriate warning code and warning message

“ERR:<errorcode>:<errormessage>” -
in case it failed, providing an appropriate error code and error message

Example:

C: “DESTROY AUDIO_OUTPUT_DEVICE Alsa”
S: “OK”

3.1.1 Getting all loaded audio output drivers

Use the following command to list all currently loaded audio output drivers, means all created audio output devices:

GET AUDIO_OUTPUT_DEVICES

Possible Answers:

LinuxSampler will answer by sending comma separated names of all created audio output devices.

Examples:

C: “GET AUDIO_OUTPUT_DEVICES”
S: “Jack”

C: “GET AUDIO_OUTPUT_DEVICES”

S: “Alsa,Jack”

3.1.1 Getting current settings of an audio output driver

Use the following command to get current settings of a specific, loaded audio output driver:

GET AUDIO_OUTPUT_DEVICE INFO <audio-output-type>

Where <audio-output-type> is the name of the audio output driver given by the “GET AVAILABLE_AUDIO_OUTPUT_TYPES” command.

Possible Answers:

LinuxSampler will answer by sending a <CRLF> separated list. Each answer line begins with the information category name followed by a colon and then a space character <SP> and finally the info character string to that info category. As some parameters might allow multiple values, character strings are encapsulated into apostrophes ('). At the moment the following information categories are defined (independently of driver):

CHANNELS –
amount of audio output channels this driver currently offers

SAMPLERATE –
playback sample rate the device uses

ACTIVE -
either true or false, if false then the audio driver is inactive and doesn't output any sound, nor do the sampler channels connected to this audio device render any audio

The mentioned fields above don't have to be in particular order. The fields above are only those fields which are returned by all audio output drivers. Every audio output driver might have its own, additional driver specific parameters (see “GET AUDIO_OUTPUT_TYPE INFO” command) which are also returned by this command.

Example:

C: “GET AUDIO_OUTPUT_DEVICE INFO Alsa”
S: “CHANNELS: 2”
“SAMPLERATE: 44100”
“ACTIVE: true”

“FRAGMENTS: 2”
“FRAGMENTSIZE: 128”
“CARD: '0,0'”
“.”

3.1.1 Changing settings of audio output drivers

Use the following command to alter a specific setting of a created audio output device:

SET AUDIO_OUTPUT_DEVICE_PARAMETER <audio-type> <key> <value>

or

SET AUDIO_OUTPUT_DEVICE_PARAMETER <audio-type> <key>=<value>

Where <audio-type> should be replaced by the name of the audio device, <key> by the name of the parameter and <value> by the new value for this parameter.

Possible Answers:

“OK” -
in case setting was successfully changed

“WRN:<warningcode>:<warningmessage>” -
in case setting was cahnged successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

“ERR:<errorcode>:<errormessage>” -
in case it failed, providing an appropriate error code and error message

Examples:

C: “SET AUDIO_OUTPUT_DEVICE_PARAMETER Alsa FRAGMENTSIZE 128”
S: “OK”

C: “SET AUDIO_OUTPUT_DEVICE_PARAMETER Alsa FRAGMENTSIZE=128”
S: “OK”

3.1.2 Getting informations about an audio channel

Use the following command to get informations about an audio channel:

GET AUDIO_OUTPUT_CHANNEL INFO <audio-output-type> <audio-chan>

Where <audio-output-type> is the name of the audio output driver and <audio-chan> the audio audio channel number.

Possible Answers:

LinuxSampler will answer by sending a <CRLF> separated list. Each answer line begins with the information category name followed by a colon and then a space character <SP> and finally the info character string to that info category. At the moment the following information categories are defined:

NAME –
arbitrary character string naming the channel
(always returned by all audio channels)

IS_MIX_CHANNEL –
either true or false, a mixchannel is not a real, independent audio channel, but a virtual channel which is mixed to another real channel, this mechanism is needed for sampler engines which need more audio channels than the used audio system might be able to offer
(always returned by all audio channels)

MIX_CHANNEL_DESTINATION -
reflects the real audio channel (of the same audio output device) this mix channel refers to, means where the audio signal actually will be routed / added to
(only returned in case the audio channel is mix channel)

The mentioned fields above don't have to be in particular order. The fields above are only those fields which are generally returned for the described cases by all audio channels regardless of the audio driver. Every audio channel might have its own, additional driver & channel specific parameters.

Examples:

C: “GET AUDIO_OUTPUT_CHANNEL INFO Alsa 1”
S: “NAME: studio monitor left”
“IS_MIX_CHANNEL: false”
“.”

C: “GET AUDIO_OUTPUT_CHANNEL INFO Alsa 3”
S: “NAME: studio monitor left”
“IS_MIX_CHANNEL: true”
“MIX_CHANNEL_DESTINATION: 1”
“.”

```
C: “GET AUDIO_OUTPUT_CHANNEL INFO Jack 1”
S: “NAME: 'ardour (left)'”
   “IS_MIX_CHANNEL: false”
   “JACK_BINDINGS: 'ardour:0'
   “.”
```

3.1.1 Getting informations about specific audio channel parameter

Use the following command to get detailed informations about a specific audio channel parameter:

GET AUDIO_OUTPUT_CHANNEL_PARAMETER INFO <audio-t> <chan> <param>

Where <audio-t> is the name of the audio output device as returned by the “GET AVAILABLE_AUDIO_OUTPUT_TYPES” command, <chan> the audio channel number and <param> a specific channel parameter name for which information should be obtained (as returned by the “GET AUDIO_OUTPUT_CHANNEL INFO” command).

Possible Answers:

LinuxSampler will answer by sending a <CRLF> separated list. Each answer line begins with the information category name followed by a colon and then a space character <SP> and finally the info character string to that info category. There are informations which are always returned, independently of the given channel parameter and there are optional informations which are only shown dependently to the given audio channel. At the moment the following information categories are defined:

DESCRIPTION –
arbitrary text describing the purpose of the parameter (always returned)

FIX -
either true or false, if true then this parameter is read only / cannot be altered (always returned)

MULTIPLICITY -
either true or false, defines if this parameter allows only one value or a list of values, where true means multiple values and false only a single value allowed (always returned)

RANGE_MIN –
defines lower limit of the allowed value range for this

parameter, can be an integer value as well as a dotted number
(optionally returned, dependent to driver & channel parameter, but always in conjunction with RANGE_MAX)

RANGE_MAX –
defines upper limit of the allowed value range for this parameter, can be an integer value as well as a dotted number
(optionally returned, dependent to driver & channel parameter, but always in conjunction with RANGE_MIN

POSSIBILITES –
comma separated list of possible values for this parameter, character strings are encapsulated into apostrophes
(optionally returned, dependent to driver & channel parameter)

The mentioned fields above don't have to be in particular order.

Example:

C: “GET AUDIO_OUTPUT_CHANNEL_PARAMETER INFO Jack 0 JACK_BINDINGS”
S: “DESCRIPTION: bindings to other Jack clients”
“FIX: false”
“MULTIPLICITY: true”
“POSSIBILITES: 'PCM:0','PCM:1','ardour:0','ardour:1'”
“.”

3.1.1 Changing settings of audio output channels

Use the following command to alter a specific setting of audio output channel:

SET AUDIO_OUTPUT_CHANNEL_PARAMETER <audio-t> <chn> <key> <value>

or

SET AUDIO_OUTPUT_CHANNEL_PARAMETER <audio-t> <chn> <key>=<value>

Where <audio-t> should be replaced by the name of the audio device, <chn> by the audio channel number, <key> by the name of the parameter and <value> by the new value for this parameter.

Possible Answers:

“OK” -

in case setting was successfully changed

"WRN:<warningcode>:<warningmessage>" -
in case setting was cahnged successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<errorcode>:<errormessage>" -
in case it failed, providing an appropriate error code and error message

Examples:

C: "SET AUDIO_OUTPUT_CHANNEL PARAMETER Jack 0 JACK_BINDINGS 'PCM:0'"
S: "OK"

C: "SET AUDIO_OUTPUT_CHANNEL PARAMETER Jack 0 JACK_BINDINGS='PCM:0'"
S: "OK"

C: "SET AUDIO_OUTPUT_CHANNEL PARAMETER Jack 0 NAME 'monitor left'"
S: "OK"

3.2 Configuring MIDI input drivers

Drivers in LinuxSampler are called devices. You can use multiple devices simultaniously, e.g. to use MIDI over ethernet as MIDI input on one sampler channel and Alsa as MIDI input on another sampler channel. Usually these devices will be created automatically by LinuxSampler when you select an MIDI input type on a sampler channel and the respective device was not created yet, but this is not always possible, because some drivers might need explicit parameters at creation time.

Instead of defining commands and parameters for each driver individually, all possible parameters, their meanings and possible values have to be obtained at runtime. This makes the protocol a bit abstract, but has the advantage, that frontends can be written independently of what drivers are implemented and what parameters these drivers are actually offering. Commands for configuring MIDI input devices are pretty much the same as the commands for configuring audio output drivers, already described in the last chapter.

3.2.1 Getting all available MIDI input drivers

Use the following command to list all MIDI input drivers currently available for the LinuxSampler instance:

GET AVAILABLE_MIDI_INPUT_TYPES

Possible Answers:

LinuxSampler will answer by sending comma separated character strings symbolizing the available MIDI input drivers.

Example:

C: “GET AVAILABLE_MIDI_INPUT_TYPES”
S: “Alsa,Jack”

3.2.1 Getting informations about a specific MIDI input driver

Use the following command to get detailed informations about a specific MIDI input driver:

GET MIDI_INPUT_TYPE INFO <midi-input-type>

Where <midi-input-type> is the name of the MIDI input driver.

Possible Answers:

LinuxSampler will answer by sending a <CRLF> separated list. Each answer line begins with the information category name followed by a colon and then a space character <SP> and finally the info character string to that info category. At the moment the following information categories are defined:

DESCRIPTION –
arbitrary description text about the MIDI input driver

VERSION -
arbitrary character string regarding the driver's version

PARAMETERS –
comma separated list of all parameters available for the given MIDI input driver

The mentioned fields above don't have to be in particular order.

Example:

C: “GET MIDI_INPUT_TYPE INFO Alsa”
S: “DESCRIPTION: Advanced Linux Sound Architecture”
“VERSION: 1.0”

“PARAMETERS: ALSA_SEQ_BINDINGS”
“.”

3.2.1 Getting informations about specific MIDI input driver parameter

Use the following command to get detailed informations about a specific parameter of a specific MIDI input driver:

GET MIDI_INPUT_TYPE_PARAMETER INFO <midi-t> <param> [<deplist>]

Where <midi-t> is the name of the MIDI input driver, <param> a specific parameter this driver offers.

Where <midi-t> is the name of the MIDI input driver as returned by the “GET AVAILABLE_MIDI_INPUT_TYPES” command, <param> a specific parameter name for which information should be obtained (as returned by the “GET MIDI_INPUT_TYPE INFO” command) and <deplist> is an optional list of parameters on which the sought parameter <param> depends on, <deplist> is a key-value pair list in form of “key1=val1 key2=val2 ...”, where character string values are encapsulated into apostrophes ('). Arguments given with <deplist> which are not dependency parameters of <param> will be ignored, means the frontend application can simply put all parameters in <deplist> with the values selected by the user.

Possible Answers:

LinuxSampler will answer by sending a <CRLF> separated list. Each answer line begins with the information category name followed by a colon and then a space character <SP> and finally the info character string to that info category. There are informations which are always returned, independent of the given driver parameter and there are optional informations which are only shown dependent to given driver parameter. At the moment the following information categories are defined:

DESCRIPTION –
arbitrary text to describe the purpose of the parameter (always returned, no matter which driver parameter)

MANDATORY -
either true or false, defines if this parameter must be given when the device is to be created by the 'CREATE MIDI_INPUT_DEVICE' command (always returned, no matter which driver parameter)

FIX -
either true or false, defines if this parameter can be

changed at any time, once the device is created by the 'CREATE MIDI_INPUT_DEVICE' command
(always returned, no matter which driver parameter)

MULTIPLICITY -
either true or false, defines if this parameter allows only one value or a list of values, where true means multiple values and false only a one value allowed
(always returned, no matter which driver parameter)

DEPENDS -
comma separated list of paramters this parameter depends on, means the values for fields 'DEFAULT', 'RANGE_MIN', 'RANGE_MAX' and 'POSSIBILITIES' might depend on these listed parameters
(optionally returned, dependent to driver parameter)

DEFAULT –
reflects the default value for this parameter which is used when the device is created and not explicitly defined with the 'CREATE MIDI_INPUT_DEVICE' command, in case of MULTIPLCITY=true, this is a comma separated list, that's why character strings are encapsulated into apostrophes (')
(optional returned, dependent to driver parameter)

RANGE_MIN –
defines lower limit of the allowed value range for this parameter, can be an integer value as well as a dotted number
(optional returned, dependent to driver parameter, but always in conjunction with RANGE_MAX)

RANGE_MAX –
defines upper limit of the allowed value range for this parameter, can be an integer value as well as a dotted number
(optional returned, dependent to driver parameter, but always in conjunction with RANGE_MIN)

POSSIBILITES –
comma separated list of possible values for this parameter, character strings are encapsulated into
(optional returned, dependent to driver parameter)

The mentioned fields above don't have to be in particular order.

Example:

C: “GET MIDI_INPUT_TYPE_PARAMETER INFO Alsa ALSA_SEQ_BINDINGS”
S: “DESCRIPTION: Bindings to other Alsa sequencer clients”
“MANDATORY: false”
“FIX: false”
“MULTIPLICITY: true”
“DEFAULT: 'NULL'”
“POSSIBILITES: 'NULL','64:0,'68:0','68:1'”
“.”

3.2.1 Loading an MIDI input driver

Use the following command to create a new MIDI input device for the desired MIDI input system:

CREATE MIDI_INPUT_DEVICE <midi-input-type> [<param-list>]

Where <midi-input-type> should be replaced by the desired MIDI input system and <param-list> by an optional list of driver specific parameters in form of “key1=val1 key2=val2 ...”, where character string values should be encapsulated into apostrophes ('). Note that there might be drivers which require parameter(s) to be given with this command. Use the previously described commands in this chapter to get those informations.

Possible Answers:

“OK” -
in case the driver was successfully loaded

“WRN:<warningcode>:<warningmessage>” -
in case the driver was loaded successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

“ERR:<errorcode>:<errormessage>” -
in case it failed, providing an appropriate error code and error message

Example:

C: “CREATE MIDI_INPUT_DEVICE Alsa”
S: “OK”

3.2.1 Unloading an MIDI input driver

Use the following command to destroy a created MIDI input device:

DESTROY MIDI_INPUT_DEVICE <midi-input-type>

Where <midi-input-type> should be replaced by the midi input system.

Possible Answers:

"OK" -
in case the driver was successfully unloaded

"WRN:<warningcode>:<warningmessage>" -
in case the driver was loaded successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<errorcode>:<errormessage>" -
in case it failed, providing an appropriate error code and error message

Example:

C: "DESTROY MIDI_INPUT_DEVICE Alsa"
S: "OK"

3.2.1 Getting all loaded MIDI input drivers

Use the following command to list all currently loaded MIDI input drivers, means all created MIDI input devices:

GET MIDI_INPUT_DEVICES

Possible Answers:

LinuxSampler will answer by sending comma separated names of all created MIDI input devices.

Examples:

C: "GET MIDI_INPUT_DEVICES"
S: "ALSA"

C: "GET MIDI_INPUT_DEVICES"
S: "Alsa,Jack"

3.2.1 Getting current settings of a MIDI input driver

Use the following command to get current settings of a specific,

loaded MIDI input driver:

GET MIDI_INPUT_DEVICE INFO <midi-input-type>

Where <midi-input-type> is the name of the MIDI input driver.

Possible Answers:

LinuxSampler will answer by sending a <CRLF> separated list. Each answer line begins with the information category name followed by a colon and then a space character <SP> and finally the info character string to that info category. As some parameters might allow multiple values, character strings are encapsulated into apostrophes ('). At the moment the following information categories are defined (independent of driver):

ACTIVE -
either true or false, if false then the MIDI driver is inactive and doesn't listen to any incoming MIDI events and thus doesn't forward them to connected sampler channels

The field above is only the one which is returned by all MIDI input drivers. Every MIDI input driver might have its own, additional driver specific parameters which are also returned by this command.

Example:

C: “GET MIDI_INPUT_DEVICE INFO Alsa”
S: “ACTIVE: true”
“.”

3.2.2 Changing settings of audio output drivers

Use the following command to alter a specific setting of a created MIDI input device:

SET MIDI_INPUT_DEVICE_PARAMETER <midi-type> <key> <value>

or

SET MIDI_INPUT_DEVICE_PARAMETER <midi-type> <key>=<value>

Where <midi-type> should be replaced by the name of the MIDI input device, <key> by the name of the parameter and <value> by the new value for this parameter.

Possible Answers:

“OK” -
in case setting was successfully changed

“WRN:<warningcode>:<warningmessage>” -
in case setting was cahnged successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

“ERR:<errorcode>:<errormessage>” -
in case it failed, providing an appropriate error code and error message

Examples:

C: “SET MIDI_INPUT_DEVICE PARAMETER Alsa ACTIVE false”
S: “OK”

C: “SET MIDI_INPUT_DEVICE PARAMETER Alsa ACTIVE=false”
S: “OK”

3.2.3 Getting informations about a MIDI port

Use the following command to get informations about a MIDI port:

GET MIDI_INPUT_PORT INFO <midi-input-type> <midi-port>

Where <midi-input-type> is the name of the MIDI inpupt driver and <midi-port> the MIDI input port number.

Possible Answers:

LinuxSampler will answer by sending a <CRLF> separated list. Each answer line begins with the information category name followed by a colon and then a space character <SP> and finally the info character string to that info category. At the moment the following information categories are defined:

NAME –
arbitrary character string naming the port

The field above is only the one which is returned by all MIDI ports regardless of the MIDI driver & port. Every MIDI port might have its own, additional driver & port specific parameters.

Example:

C: “GET MIDI_INPUT_PORT INFO Alsa 0”
S: “NAME: Masterkeyboard”
“ALSA_SEQ_BINDINGS: '64:0'”
“.”

3.2.1 Getting informations about specific MIDI port parameter

Use the following command to get detailed informations about a specific MIDI port parameter:

GET MIDI_INPUT_PORT_PARAMETER INFO <midi-t> <port> <param>

Where <midi-t> is the name of the MIDI input device as returned by the “GET AVAILABLE_MIDI_INPUT_TYPES” command, <port> the MIDI port number and <param> a specific port parameter name for which information should be obtained (as returned by the “GET MIDI_INPUT_PORT INFO” command).

Possible Answers:

LinuxSampler will answer by sending a <CRLF> separated list. Each answer line begins with the information category name followed by a colon and then a space character <SP> and finally the info character string to that info category. There are informations which are always returned, independently of the given channel parameter and there are optional informations which are only shown dependently to the given MIDI port. At the moment the following information categories are defined:

DESCRIPTION –
arbitrary text describing the purpose of the parameter (always returned)

FIX -
either true or false, if true then this parameter is read only / cannot be altered (always returned)

MULTIPLICITY -
either true or false, defines if this parameter allows only one value or a list of values, where true means multiple values and false only a single value allowed (always returned)

RANGE_MIN –
defines lower limit of the allowed value range for this parameter, can be an integer value as well as a dotted

number
(optionally returned, dependent to driver & port parameter, but always in conjunction with RANGE_MAX)

RANGE_MAX –
defines upper limit of the allowed value range for this parameter, can be an integer value as well as a dotted number
(optionally returned, dependent to driver & port parameter, but always in conjunction with RANGE_MIN

POSSIBILITES –
comma separated list of possible values for this parameter, character strings are encapsulated into apostrophes
(optionally returned, dependent to driver & port parameter)

The mentioned fields above don't have to be in particular order.

Example:

C: “GET MIDI_INPUT_PORT_PARAMETER INFO Alsa 0 ALSA_SEQ_BINDINGS”
S: “DESCRIPTION: bindings to other Alsa sequencer clients”
“FIX: false”
“MULTIPLICITY: true”
“POSSIBILITES: '64:0','68:0','68:1'”
“.”

3.2.1 Changing settings of MIDI input ports

Use the following command to alter a specific setting of a MIDI input port:

SET MIDI_INPUT_PORT PARAMETER <midi-t> <port> <key> <value>

or

SET MIDI_INPUT_PORT PARAMETER <midi-t> <port> <key>=<value>

Where <midi-t> should be replaced by the name of the MIDI device, <port> by the MIDI port number, <key> by the name of the parameter and <value> by the new value for this parameter.

Possible Answers:

“OK” -
in case setting was successfully changed

“WRN:<warningcode>:<warningmessage>” -
in case setting was cahnged successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

“ERR:<errorcode>:<errormessage>” -
in case it failed, providing an appropriate error code and error message

Examples:

C: “SET MIDI_INPUT_PORT PARAMETER Alsa 0 ALSA_SEQ_BINDINGS 'PCM:0'”
S: “OK”

C: “SET MIDI_INPUT_PORT PARAMETER Alsa 0 ALSA_SEQ_BINDINGS='PCM:0'”
S: “OK”

C: “SET MIDI_INPUT_PORT PARAMETER Alsa 0 NAME='My Masterkeyboard'”
S: “OK”

3.3 Configuring sampler channels

The following commands describe how to add and remove sampler channels, deploy sampler engines, load instruments and connect sampler channels to MIDI and audio devices.

3.3.1 Loading an instrument

An instrument file can be loaded and assigned to a sampler channel by the following command:

LOAD INSTRUMENT <filename> <instr-index> <sampler-channel>

Where <filename> is the name of the instrument file on the LinuxSampler instance's host system, <instr-index> the index of the instrument in the instrument file and <sampler-channel> is the number of the sampler channel the instrument should be assigned to. Each sampler channel can only have one instrument.

Possible Answers:

“OK” -
in case the instrument was successfully loaded

“WRN:<warningcode>:<warningmessage>” -

in case the instrument was loaded successfully, but there are noteworthy issue(s) related (e.g. Engine doesn't support one or more patch parameters provided by the loaded instrument file), providing an appropriate warning code and warning message

"ERR:<errorcode>:<errormessage>" -
in case it failed, providing an appropriate error code and error message

3.3.2 Loading a sampler engine

A sample engine can be deployed and assigned to a specific sampler channel by the following command:

```
LOAD ENGINE <engine-name> <sampler-channel>
```

Where <engine-name> is usually the C++ class name of the engine implementation and <sampler-channel> the sampler channel the deployed engine should be assigned to. Even if the respective sampler channel has already a deployed engine with that engine name, a new engine instance will be assigned to the sampler channel.

Possible Answers:

"OK" -
in case the engine was successfully deployed

"WRN:<warningcode>:<warningmessage>" -
in case the engine was deployed successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<errorcode>:<errormessage>" -
in case it failed, providing an appropriate error code and error message

3.3.3 Current number of sampler channels

The number of sampler channels can change on runtime. To get the current amount of sampler channels, the frontend can send the following command:

```
GET CHANNELS
```

Possible Answers:

LinuxSampler will answer returning the number of channels.

Example:

C: “GET CHANNELS”
S: “32”

3.3.4 Adding a new sampler channel

A new sampler channel can be added to the end of the sampler channel list by sending the following command:

ADD CHANNEL

This will increment the sampler channel count by one and the new sampler channel will be appended to the end of the sampler channel list. The frontend should send the respective, related commands right after to e.g. load an engine, load an instrument and setting input, output method and evtl. other commands to initialize the new channel. The frontend should use the sampler channel returned by the answer of this command to perform the previously recommended commands, to avoid race conditions e.g. with other frontends that might also have sent an “ADD CHANNEL” command.

Possible Answers:

“OK[<sampler-channel>]” -
in case a new sampler channel could be added, where <sampler-channel> reflects the channel number of the new created sampler channel which should the be used to set up the sampler channel by sending subsequent intialization commands

“WRN:<warningcode>:<warningmessage>” -
in case a new channel was added succesfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

“ERR:<errorcode>:<errormessage>” -
in case it failed, providing an appropriate error code and error message

3.3.5 Removing a sampler channel

A sampler channel can be removed by sending the following command:

REMOVE CHANNEL <sampler-channel>

This will decrement the sampler channel count by one and also decrement the channel numbers of all subsequent sampler channels by one.

Possible Answers:

"OK" -
in case the given sampler channel could be removed

"WRN:<warningcode>:<warningmessage>" -
in case the given channel was removed, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<errorcode>:<errormessage>" -
in case it failed, providing an appropriate error code and error message

3.3.6 Getting all available engines

The frontend can ask for all available engines by sending the following command:

```
GET AVAILABLE_ENGINES
```

Possible Answers:

LinuxSampler will answer by sending a comma separated character string of the engines' C++ class names.

Example:

```
C: "GET AVAILABLE_ENGINES"
S: "GigEngine,AkaiEngine,DLSEngine,JoesCustomEngine"
```

3.3.7 Getting informations about an engine

The frontend can ask for informations about a specific engine by sending the following command:

```
GET ENGINE INFO <engine-name>
```

Where <engine-name> is usually the C++ class name of the engine implementation.

Possible Answers:

LinuxSampler will answer by sending a <CRLF> separated list. Each answer line begins with the information category name followed by a colon and then a space character <SP> and finally the info character string to that info category. At the moment the following categories are defined:

DESCRIPTION –
arbitrary description text about the engine

VERSION -
arbitrary character string regarding the engine's version

The mentioned fields above don't have to be in particular order.

Example:

C: “GET ENGINE INFO JoesCustomEngine”
S: “DESCRIPTION: this is Joe's custom sampler engine”
“VERSION: testing-1.0”
“.”

3.3.8 Getting sampler channel informations

The frontend can ask for the current settings of a sampler channel by sending the following command:

GET CHANNEL INFO <sampler-channel>

Where <sampler-channel> is the sampler channel number the frontend is interested in.

Possible Answers:

LinuxSampler will answer by sending a <CRLF> separated list. Each answer line begins with the settings category name followed by a colon and then a space character <SP> and finally the info character string to that setting category. At the moment the following categories are defined:

ENGINE_NAME –
name of the engine that is deployed on the sampler channel, “<NONE>” if there's no engine deployed yet for this sampler channel

AUDIO_OUTPUT_TYPE –

output system which is currently used to output the audio signal (at the moment either “ALSA” or “JACK”)

AUDIO_OUTPUT_CHANNELS –
number of output channels the sampler channel offers (dependent to used sampler engine and loaded instrument)

AUDIO_OUTPUT_ROUTING -
comma separated list which reflects to which audio channel of the selected audio output device each sampler output channel is routed to, e.g. “0,3” would mean the engine's output channel 0 is routed to channel 0 of the audio output device and the engines's output channel 1 is routed to the channel 3 of the audio output device

INSTRUMENT_FILE –
the file name of the loaded instrument, “<NONE>” if there's no instrument yet loaded for this sampler channel

INSTRUMENT_NR -
the instrument index number of the loaded instrument

MIDI_INPUT_TYPE –
at the moment only “ALSA”, but will change in future

MIDI_INPUT_PORT –
port number of the MIDI input device

MIDI_INPUT_CHANNEL –
the MIDI input channel number this sampler channel should listen to or ALL to listen on all MIDI channels

VOLUME –
optionally dotted number for the channel volume factor (where a value < 1.0 means attenuation and a value > 1.0 means amplification)

The mentioned fields above don't have to be in particular order.

Example:

C: “GET CHANNEL INFO 34”
S: “ENGINE_NAME: GigEngine”
“VOLUME: 1.0”
“AUDIO_OUTPUT_TYPE: ALSA”
“AUDIO_OUTPUT_CHANNELS: 2”
“AUDIO_OUTPUT_ROUTING: 0,1”

```
"INSTRUMENT_FILE: /home/joe/FazioliPiano.gig"
"INSTRUMENT_NR: 0"
"MIDI_INPUT_TYPE: ALSA"
"MIDI_INPUT_PORT: 0"
"MIDI_INPUT_CHANNEL: 5"
"."
```

3.3.9 Current number of active voices

The frontend can ask for the current number of active voices on a sampler channel by sending the following command:

```
GET CHANNEL VOICE_COUNT <sampler-channel>
```

Where <sampler-channel> is the sampler channel number the frontend is interested in.

Possible Answers:

LinuxSampler will answer by returning the number of active voices on that channel.

3.3.10 Current number of active disk streams

The frontend can ask for the current number of active disk streams on a sampler channel by sending the following command:

```
GET CHANNEL STREAM_COUNT <sampler-channel>
```

Where <sampler-channel> is the sampler channel number the frontend is interested in.

Possible Answers:

LinuxSampler will answer by returning the number of active disk streams on that channel in case the engine supports disk streaming, if the engine doesn't support disk streaming it will return "NA" for not available.

3.3.11 Current fill state of disk stream buffers

The frontend can ask for the current fill state of all disk streams on a sampler channel by sending the following command:

```
GET CHANNEL BUFFER_FILL BYTES <sampler-channel>
```

to get the fill state in bytes or

GET CHANNEL BUFFER_FILL PERCENTAGE <sampler-channel>

to get the fill state in percent, where <sampler-channel> is the sampler channel number the frontend is interested in.

Possible Answers:

LinuxSampler will either answer by returning a comma separated string with the fill state of all disk stream buffers on that channel or an empty line if there are no active disk streams or “NA” for *not available* in case the engine which is deployed doesn't support disk streaming. Each entry in the answer list will begin with the stream's ID in brackets followed by the numerical representation of the fill size (either in bytes or percentage). Note: due to efficiency reasons the fill states in the response are not in particular order, thus the frontend has to sort them by itself if necessary.

Example:

C: “GET CHANNEL BUFFER_FILL BYTES 4”
S: “[115]420500,[116]510300,[75]110000,[120]230700”

C: “GET CHANNEL BUFFER_FILL PERCENTAGE 4”
S: “[115]90%,[116]98%,[75]40%,[120]62%”

C: “GET CHANNEL BUFFER_FILL PERCENTAGE 4”
S: “”

3.3.12 Setting audio output type

The frontend can alter the audio output type on a specific sampler channel by sending the following command:

SET CHANNEL AUDIO_OUTPUT_TYPE <sampler-channel> <audio-output-type>

Where <audio-output-type> is currently either “ALSA” or “JACK” and <sampler-channel> is the respective sampler channel number.

Possible Answers:

“OK” -
on success

“WRN:<warningcode>:<warningmessage>” -
if audio output type was set, but there are noteworthy

issue(s) related, providing an appropriate warning code and warning message

“ERR:<errorcode>:<errormessage>” -
in case it failed, providing an appropriate error code and error message

3.3.13 Setting audio output channel

The frontend can alter the audio output channel on a specific sampler channel by sending the following command:

SET CHANNEL AUDIO_OUTPUT_CHANNEL <sampler-chan> <audioout> <audioin>

Where <sampler-chan> is the sampler channel, <audioout> is the sampler channel's audio output channel which should be rerouted and <audioin> the audio channel of the selected audio output device where <audioout> should be routed to.

Possible Answers:

“OK” -
on success

“WRN:<warningcode>:<warningmessage>” -
if audio output channel was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

“ERR:<errorcode>:<errormessage>” -
in case it failed, providing an appropriate error code and error message

3.3.14 Setting MIDI input port

The frontend can alter the input MIDI port on a specific sampler channel by sending the following command:

SET CHANNEL MIDI_INPUT_PORT <sampler-channel> <midi-input-port>

Where <midi-input-port> is a MIDI input port number of the MIDI input device connected to the sampler channel given by <sampler-channel>.

Possible Answers:

“OK” -

on success

"WRN:<warningcode>:<warningmessage>" -
if MIDI input port was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<errorcode>:<errormessage>" -
in case it failed, providing an appropriate error code and error message

3.3.15 Setting MIDI input channel

The frontend can alter the MIDI channel a sampler channel should listen to by sending the following command:

SET CHANNEL MIDI_INPUT_CHANNEL <sampler-channel> <midi-input-chan>

Where <midi-input-chan> is the new MIDI input channel where <sampler-channel> should listen to or ALL to listen on all 16 MIDI channels.

Possible Answers:

"OK" -
on success

"WRN:<warningcode>:<warningmessage>" -
if MIDI input channel was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<errorcode>:<errormessage>" -
in case it failed, providing an appropriate error code and error message

3.3.16 Setting channel volume

The frontend can alter the volume of a sampler channel by sending the following command:

SET CHANNEL VOLUME <sampler-channel> <volume>

Where <volume> is an optionally dotted positive number (a value smaller than 1.0 means attenutation, whereas a value greater than 1.0 means amplification) and <sampler-channel> defines the sampler channel where this volume factor should be set.

Possible Answers:

"OK" -
on success

"WRN:<warningcode>:<warningmessage>" -
if channel volume was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<errorcode>:<errormessage>" -
in case it failed, providing an appropriate error code and error message

3.3.17 Resetting a sampler channel

The frontend can reset a particular sampler channel by sending the following command:

RESET CHANNEL <sampler-channel>

Where <sampler-channel> defines the sampler channel to be reset. This will cause the engine on that sampler channel, its voices and eventually disk streams and all control and status variables to be reset.

Possible Answers:

"OK" -
on success

"WRN:<warningcode>:<warningmessage>" -
if channel was reset, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<errorcode>:<errormessage>" -
in case it failed, providing an appropriate error code and error message

3.3.18 Register frontend for receiving UDP event messages

The frontend can register itself to the LinuxSampler application to be informed about noteworthy events by sending this command:

SUBSCRIBE NOTIFICATION <udp-port>

Where <udp-port> is the UDP port number on the frontend's host on which the frontend will listen to. The frontend has to open, listen and react on that port before it tries to register itself for NOTIFICATION, because the LinuxSampler instance will send a PING packet to test if the UDP is actually reachable and the frontend is listening on that port. The frontend will then immediately have to answer by sending a PONG packet, else the SUBSCRIBE NOTIFICATION command will fail (see UDP chapter for PING and PONG packets). The LinuxSampler instance will periodically send PING packets on which the frontend has to answer, else LinuxSampler assumes the frontend to be not available and will stop to send notification / event messages.

Possible Answers:

"OK[<session-id>]" -
    on success, where <session-id> will be replaced by a character string reflecting the ID needed for unsubscription

"WRN:<warningcode>:<warningmessage>" -
    if registration succeeded, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<errorcode>:<errormessage>" -
    in case it failed, providing an appropriate error code and error message

3.3.19 Deregister frontend for not receiving UDP event messages anymore

The frontend can deregister itself if it doesn't want to receive UDP event packets anymore by sending the following command:

UNSUBSCRIBE NOTIFICATION <session-id>

Where <session-id> should be replaced by the ID returned from the "SUBSCRIBE NOTIFICATION" command (see 3.17).

Possible Answers:

"OK" -
    on success

"WRN:<warningcode>:<warningmessage>" -
    if deregistration succeeded, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

“ERR:<errorcode>:<errormessage>” -
in case it failed, providing an appropriate error code and error message

3.3.20 Close client connection

The client can close its network connection to LinuxSampler by sending the following command:

QUIT

This is probably more interesting for manual telnet connections to LinuxSampler than really useful for a frontend implementation.

4. Command Syntax

The following are the LSCP (LinuxSampler control protocol) commands:

ADD <SP> CHANNEL

GET <SP> <get-instruction>

LOAD <SP> <load-instruction>

REMOVE <SP> CHANNEL <SP> <sampler-channel>

SET <SP> CHANNEL <SP> <set-chan-instruction>

RESET <SP> CHANNEL <SP> <sampler-channel>

SUBSCRIBE <SP> NOTIFICATION <SP> <udp-port>

UNSUBSCRIBE <SP> NOTIFICATION <SP> <session-id>

QUIT

The syntax of the above argument fields is given below using Backus-Naur Form (BNF as described in RFC-2234 [2]) where applicable.

<get-instruction> ::=
AVAILABLE_ENGINES |
CHANNELS |
CHANNEL <SP> INFO <SP> <sampler-channel> |

```
            CHANNEL <SP> BUFFER_FILL <SP> <buffer-size-type> <SP>
                <sampler-channel> |
            CHANNEL <SP> STREAM_COUNT <SP> <sampler-channel> |
            CHANNEL <SP> VOICE_COUNT <SP> <sampler-channel> |
            ENGINE <SP> INFO <SP> <engine-name>

<load-instruction> ::=
            INSTRUMENT <SP> <load-instr-args> |
            ENGINE <SP> <load-engine-args>

<sampler-channel> ::= <number>

<set-chan-instruction> ::=
            AUDIO_OUTPUT_CHANNEL <SP> <sampler-channel> <SP>
                <audio-output-channel> |
            AUDIO_OUTPUT_TYPE <SP> <sampler-channel> <SP>
                <audio-output-type> |
            MIDI_INPUT_PORT <SP> <sampler-channel> <SP>
                <midi-input-port> |
            MIDI_INPUT_CHANNEL <SP> <sampler-channel> <SP>
                <midi-input-channel> |
            MIDI_INPUT_TYPE <SP> <sampler-channel> <SP>
                <midi-input-type> |
            VOLUME <SP> <sampler-channel> <SP> <volume>

<udp-port> ::= <number>

<session-id> ::= <string>

<buffer-size-type> ::= BYTES | PERCENTAGE

<engine-name> ::= <cpp-classname>

<load-instr-args> ::=
        <filename> <SP> <instr-index> <SP> <sampler-channel>

<load-engine-args> ::= <engine-name> <SP> <sampler-channel>

<audio-output-channel> ::= <number>

<audio-output-type> ::= ALSA | JACK

<midi-input-port> ::= <string>

<midi-input-channel> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
                         11 | 12 | 13| 14 | 15 | 16

<midi-input-type> ::= ALSA
```

<volume> ::= <dotnum>

<cpp-classname> ::= class name as defined by the C++ programming language

<filename> ::= <string>

<string> ::= <char> | <char> <string>

<char> ::= <c> | "\" <x>

<c> ::= any one of the 128 ASCII characters, but not any <special> or <SP>

<special> ::= "<" | ">" | ";" | ":" | "&" | "{" | "}" | the control characters (ASCII codes 0 through 31 inclusive and 127)

<dotnum> ::= <snum> "." <number>

<number> ::= <d> | <d> <number>

<d> ::= any one of the ten digits 0 through 9

<snum> ::= arbitrary number of digits representing a decimal integer value in the range including 0 to infinity

<CRLF> ::= <CR> <LF>

<CR> ::= the carriage return character (ASCII code 13)

<LF> ::= the line feed character (ASCII code 10)

<SP> ::= the space character (ASCII code 32)

<x> ::= any one of the 128 ASCII characters (no exceptions)

<epsilon> ::= empty input

Note that command lines have to be <CRLF> terminated, thus the total message set / command set is defined as:

<input> ::= <epsilon> | <input> <line>

<line> ::= <CRLF> | <command> <CRLF>

where <command> is one of the command lines as defined in the

beginning of this section.

5. Events and special UDP packets

This chapter will describe all currently defined UDP packets sent by LinuxSampler.

5.1 Number of sampler channels changed

In this case LinuxSampler will send the following packet:

“CHANGE CHANNELS <channels>”

Where <channels> will be reaplaced by the new number of sampler channels.

5.2 Number of active voices changed

In this case LinuxSampler will send a packet with following shape:

CHANGE CHANNEL VOICE_COUNT <sampler-channel> <voices>

Where <sampler-channel> will be replaced by the sampler channel the voice count change occurred and <voices> by the new number of active voices on that channel.

5.3 Number of active disk streams changed

In this case LinuxSampler will send a packet with following shape:

CHANGE CHANNEL STREAM_COUNT <sampler-channel> <streams>

Where <sampler-channel> will be replaced by the sampler channel the stream count change occurred and <stream> by the new number of active disk streams on that channel.

5.4 Disk stream buffer fill state changed

In this case LinuxSampler will send a packet with following shape:

CHANGE CHANNEL BUFFER_FILL <sampler-channel>

Where <sampler-channel> will be replaced by the sampler channel the buffer fill state change occurred. The frontend will have to send

the respective command to actually get the fill state values. This is unavoidable due to the packet size limit of UDP.

5.5 Channel informations changed

In this case LinuxSampler will send a packet with following shape:

CHANGE CHANNEL INFO <sampler-channel>

Where <sampler-channel> will be replaced by the sampler channel the channel info change occurred. The frontend will have to send the respective command to actually get the channel info. This is unavoidable due to the packet size limit of UDP.

5.6 Special packet PING

Sense behind this packet is to check if the frontend is (still) listening on it's registered UDP port. This special packet has this shape:

PING <udp-port> <string>

Where <string> is an arbitrary character string that has to be confirmed by the frontend by sending a PONG UDP packet to the UDP port given by <udp-port> to LinuxSampler's host IP address.

5.7 Special packet PONG

This packet has to be returned by the frontend in reaction to a PING packet received from LinuxSampler. A PONG packet looks like this:

PONG <string>

Where <string> is a character string transmitted with PING, which should be send in order to confirm the PING packet.

Example:

S: “PING 2067 ahj_89zdiQ”
C: “PONG ahj_89zdiQ” (sent to port 2067 of LinuxSampler's host)

6. Event Syntax

The following are the defined event messages sent via UDP (only in case the frontend registered itself to receive UDP event packets):

```
CHANGE <SP> <event-arg>

PING <SP> <udp-port> <SP> <string>

PONG <SP> <string>
```

The syntax of the above argument fields is given below using Backus-Naur Form (BNF as described in RFC-2234 [3]) where applicable.

```
<event-arg> ::=
            CHANNELS <SP> <channels> |
            CHANNEL <SP> VOICE_COUNT <SP> <sampler-channel> <SP>
                <voice-count> |
            CHANNEL <SP> STREAM_COUNT <SP> <sampler-channel>
                <SP> <stream-count> |
            CHANNEL <SP> BUFFER_FILL <SP> <sampler-channel> |
            CHANNEL <SP> INFO <SP> <sampler-channel>

<udp-port> ::= <number>

<sampler-channel> ::= <number>

<string> ::= <char> | <char> <string>

<channels> ::= <number>

<voice-count> ::= <number>

<stream-count> ::= <number>

<char> ::= <c> | "\" <x>

<c> ::= any one of the 128 ASCII characters, but not any
        <special> or <SP>

<special> ::= "<" | ">" | ";" | ":" | "&" | "{" | "}" | the
              control characters (ASCII codes 0 through 31
              inclusive and 127)

<number> ::= <d> | <d> <number>
```

<d> ::= any one of the ten digits 0 through 9

<x> ::= any one of the 128 ASCII characters (no exceptions)

<SP> ::= the space character (ASCII code 32)

Security Considerations

As there is so far no method of authentication and authorisation defined and so not required for a client applications to succeed to connect, running LinuxSampler might be a security risk for the host system the LinuxSampler instance is running on.

References

< Your references will be listed here. View "Page Layout" if they are not currently visible. >

Acknowledgments

<Add any acknowledgements>

Author's Addresses

<Firstname> <Lastname>
<Affiliation>
<Address>
Phone: <optional>
Email: <Your email address>

1 Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997
2 Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997
3 Crocker, D. and Overell, P.(Editors), "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, Internet Mail Consortium and Demon Internet Ltd., November 1997