LinuxSampler Developer's
Internet Draft
Document:
draft-linuxsampler-protocol-07.sxw

Expires: July 2004

C. Schoenebeck
Interessengemeinschaft
Software Engineering
e. V.
Saturday, June 5,
2004

LinuxSampler Control Protocol

Status of this Memo

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. Focus of this protocol

Main focus of this protocol is to provide a way to configure a
running LinuxSampler instance and to retrieve informations about the
same. The focus of this protocol is not to provide a way to control
synthesis parameters or even to trigger or release notes. Or in
other words; the focus are those functionalities which are not

covered by MIDI or which may at most be handled via MIDI System
Exclusive Messages.

3.   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.

3.1   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.

3.2   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.

4.   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.

4.1   Ignored lines and comments

White lines, that is lines which only contain space and tabulator characters, and lines that start with a "#" character are ignored, thus it's possible for example to group commands and to place comments in a LSCP script file.

4.2   Configuring audio drivers

Instances of drivers in LinuxSampler are called devices. You can use multiple audio 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. For particular audio output systems it's also possible to create several devices of the same audio output driver, e.g. two separate Alsa audio output devices for using two different sound cards at the same time. This chapter describes all commands to configure LinuxSampler's audio output devices 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 currently implemented and what parameters these drivers are actually offering. This means frontends can even handle drivers which are implemented somewhere in future without modifying the frontend at all.

Note: examples in this chapter showing particular parameters of drivers are not meant as specification of the drivers' parameters. Driver implementations in LinuxSampler might have complete different parameter names and meanings than shown in these examples or might change in future, so these examples are only meant for showing how to retrieve what parameters drivers are offering, how to retrieve their possible values, etc.

4.2.1 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_DRIVERS

Possible Answers:

    LinuxSampler will answer by sending comma separated character
    strings, each symbolizing an audio output driver.

Example:

    C: "GET AVAILABLE_AUDIO_OUTPUT_DRIVERS"
    S: "ALSA,JACK"

4.2.2 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_DRIVER INFO <audio-output-driver>

Where <audio-output-driver> is the name of the audio output driver,
returned by the "GET AVAILABLE_AUDIO_OUTPUT_DRIVERS" 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_DRIVER INFO ALSA"
S: "DESCRIPTION: Advanced Linux Sound Architecture"
   "VERSION: 1.0"
   "PARAMETERS:DRIVER,CHANNELS,SAMPLERATE,ACTIVE,FRAGMENTS,
    FRAGMENTSIZE,CARD"
   "."

4.2.3 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_DRIVER_PARAMETER INFO <audio> <prm> [<deplist>]

Where <audio> is the name of the audio output driver as returned by
the "GET AVAILABLE_AUDIO_OUTPUT_DRIVERS" command, <prm> a specific
parameter name for which information should be obtained (as
returned by the "GET AUDIO_OUTPUT_DRIVER 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 already 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:

        TYPE –
            either "BOOL" for boolean value(s) or "INT" for integer
            value(s) or "FLOAT" for dotted number(s) or "STRING" for
            character string(s)
            (always returned, no matter which driver parameter)

        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, this parameter is often used in conjunction
    with RANGE_MAX, but may also appear without
    (optionally returned, dependent to driver parameter)

            RANGE_MAX –
                defines upper limit of the allowed value range for this
                parameter, can be an integer value as well as a dotted
                number, this parameter is often used in conjunction with
                RANGE_MIN, but may also appear without
                (optionally returned, dependent to driver parameter)

            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_DRIVER_PARAMETER INFO ALSA CARD"
  S: "DESCRIPTION: sound card to be used"
     "TYPE: STRING"
     "MANDATORY: false"
     "FIX: true"
     "MULTIPLICITY: false"
     "DEFAULT: '0,0'"
     "POSSIBILITES: '0,0','1,0','2,0'"
     "."

  C: "GET AUDIO_OUTPUT_DRIVER_PARAMETER INFO ALSA SAMPLERATE"
  S: "DESCRIPTION: output sample rate in Hz"
     "TYPE: INT"
     "MANDATORY: false"
     "FIX: false"
     "MULTIPLICITY: false"
     "DEPENDS: CARD"
     "DEFAULT: 44100"
     "."

  C: "GET AUDIO_OUTPUT_DRIVER_PARAMETER INFO ALSA SAMPLERATE
       CARD='0,0'"
  S: "DESCRIPTION: output sample rate in Hz"
     "TYPE: INT"
     "MANDATORY: false"
     "FIX: false"
     "MULTIPLICITY: false"
     "DEPENDS: CARD"
     "DEFAULT: 44100"
     "RANGE_MIN: 22050"
     "RANGE_MAX: 96000"
     "."

4.2.4 Creating an audio output device

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

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

    Where <audio-output-driver> 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[<device-id>]" -
            in case the device was successfully created, where
            <device-id> is the numerical ID of the new device

        "WRN[<device-id>]:<warningcode>:<warningmessage>" -
            in case the device was created successfully, where
            <device-id> is the numerical ID of the new device, but there
            are noteworthy issue(s) related (e.g. sound card doesn't
            support 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"
        S: "OK[0]"

        C: "CREATE AUDIO_OUTPUT_DEVICE ALSA CARD='2,0' SAMPLERATE=96000"
        S: "OK[1]"

4.2.5 Destroying an audio output device

    Use the following command to destroy a created output device:

        DESTROY AUDIO_OUTPUT_DEVICE <device-id>

Where <device-id> should be replaced by the numerical ID of the
audio output device as given by the "CREATE AUDIO_OUTPUT_DEVICE" or
"GET AUDIO_OUTPUT_DEVICES" command.

Possible Answers:

> "OK" -
> > in case the device was successfully destroyed

> "WRN:<warningcode>:<warningmessage>" -
> > in case the device was destroyed 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 0"
> S: "OK"

4.2.6 Getting all created audio output devices

Use the following command to list all created audio output devices:

> GET AUDIO_OUTPUT_DEVICES

Possible Answers:

> LinuxSampler will answer by sending a comma separated list with
> the numerical IDs of all audio output devices.

Examples:

> C: "GET AUDIO_OUTPUT_DEVICES"
> S: "0,1,4,5"

4.2.7 Getting current settings of an audio output device

Use the following command to get current settings of a specific,
created audio output device:

        GET AUDIO_OUTPUT_DEVICE INFO <device-id>

   Where <device-id> should be reaplced by be numerical ID of the
   audio output device as e.g. returned by the "GET
   AUDIO_OUTPUT_DEVICES" 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 device):

            DRIVER –
                identifier of the used audio output driver, as also
                returned by the "GET AVAILABLE_AUDIO_OUTPUT_DRIVERS"
                command

            CHANNELS –
                amount of audio output channels this device currently
                offers

            SAMPLERATE –
                playback sample rate the device uses

            ACTIVE -
                either true or false, if false then the audio device 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 devices. Every audio output driver
        might have its own, additional driver specific parameters (see
        "GET AUDIO_OUTPUT_DRIVER INFO" command) which are also returned
        by this command.

   Example:

       C: "GET AUDIO_OUTPUT_DEVICE INFO 0"
       S: "DRIVER: ALSA"
          "CHANNELS: 2"
          "SAMPLERATE: 44100"
          "ACTIVE: true"
          "FRAGMENTS: 2"

        "FRAGMENTSIZE: 128"
        "CARD: '0,0'"
        "."

4.2.8 Changing settings of audio output devices

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

        SET AUDIO_OUTPUT_DEVICE_PARAMETER <device-id> <key> <value>

    or alternatively:

        SET AUDIO_OUTPUT_DEVICE_PARAMETER <device-id> <key>=<value>

    Where <device-id> should be replaced by the numerical ID of the
    audio output device, <key> by the name of the parameter to change
    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 0 FRAGMENTSIZE 128"
        S: "OK"

        C: "SET AUDIO_OUTPUT_DEVICE_PARAMETER 0 FRAGMENTSIZE=128"
        S: "OK"

4.2.9 Getting informations about an audio channel

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

        GET AUDIO_OUTPUT_CHANNEL INFO <device-id> <audio-chan>

   Where <device-id> is the numerical ID of the audio output device
   and <audio-chan> the 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, which
             doesn't have to be unique
             (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 0 0"
      S: "NAME: studio monitor left"
         "IS_MIX_CHANNEL: false"
         "."

      C: "GET AUDIO_OUTPUT_CHANNEL INFO 0 1"
      S: "NAME: studio monitor right"
         "IS_MIX_CHANNEL: false"

```
    ".""

    C: "GET AUDIO_OUTPUT_CHANNEL INFO 0 2"
    S: "NAME: studio monitor left"
       "IS_MIX_CHANNEL: true"
       "MIX_CHANNEL_DESTINATION: 1"
       "."

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

4.2.10 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 <dev-id> <chan> <param>

   Where <dev-id> is the numerical ID of the audio output device as
   returned by the "GET AUDIO_OUTPUT_DEVICES" 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:

         TYPE –
             either "BOOL" for boolean value(s) or "INT" for integer
             value(s) or "FLOAT" for dotted number(s) or "STRING" for
             character string(s)
             (always returned)

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

FIX -
     either true or false, if true then this parameter is
     read only, thus 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, usually used in conjunction with 'RANGE_MAX',
     but may also appear without
     (optionally returned, dependent to driver & channel
     parameter)

RANGE_MAX –
     defines upper limit of the allowed value range for this
     parameter, can be an integer value as well as a dotted
     number, usually used in conjunction with 'RANGE_MIN',
     but may also appear without
     (optionally returned, dependent to driver & channel
     parameter)

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 1 0 JACK_BINDINGS"
S: "DESCRIPTION: bindings to other Jack clients"
   "TYPE: STRING"
   "FIX: false"
   "MULTIPLICITY: true"
   "POSSIBILITES: 'PCM:0','PCM:1','ardour:0','ardour:1'"
   "."

4.2.11 Changing settings of audio output channels

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

    SET AUDIO_OUTPUT_CHANNEL_PARAMETER <dev-id> <chn> <key> <value>

or alternatively:

    SET AUDIO_OUTPUT_CHANNEL_PARAMETER <dev-id> <chn> <key>=<value>

Where <dev-id> should be replaced by the numerical ID of the audio
device, <chn> by the audio channel number, <key> by the name of the
parameter to change 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 0 0 JACK_BINDINGS PCM:0"
S: "OK"

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

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

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

4.3   Configuring MIDI input drivers

Instances of drivers in LinuxSampler are called devices. You can
use multiple MIDI 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. For particular MIDI input systems

it's also possible to create several devices of the same MIDI input type. This chapter describes all commands to configure LinuxSampler's MIDI input devices 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 currently implemented and what parameters these drivers are actually offering. This means frontends can even handle drivers which are implemented somewhere in future without modifying the frontend at all.

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.

Note: examples in this chapter showing particular parameters of drivers are not meant as specification of the drivers' parameters. Driver implementations in LinuxSampler might have complete different parameter names and meanings than shown in these examples or might change in future, so these examples are only meant for showing how to retrieve what parameters drivers are offering, how to retrieve their possible values, etc.

4.3.1 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_DRIVERS

Possible Answers:

    LinuxSampler will answer by sending comma separated character strings, each symbolizing a MIDI input driver.

Example:

    C: "GET AVAILABLE_MIDI_INPUT_DRIVERS"
    S: "ALSA,JACK"

4.3.2 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_DRIVER INFO <midi-input-driver>

    Where <midi-input-driver> 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_DRIVER INFO ALSA"
       S: "DESCRIPTION: Advanced Linux Sound Architecture"
          "VERSION: 1.0"
          "PARAMETERS: DRIVER,ACTIVE"
          "."

4.3.3 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_DRIVER_PARAMETER INFO <midit> <param> [<deplist>]

    Where <midi-t> is the name of the MIDI input driver as returned by
    the "GET AVAILABLE_MIDI_INPUT_DRIVERS" command, <param> a specific
    parameter name for which information should be obtained (as
    returned by the "GET MIDI_INPUT_DRIVER 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:

            TYPE –
                either "BOOL" for boolean value(s) or "INT" for integer
                value(s) or "FLOAT" for dotted number(s) or "STRING" for
                character string(s)
                (always returned, no matter which driver parameter)

            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, this parameter is usually used in conjuncion
                 with 'RANGE_MAX' but may alo appear without
                 (optional returned, dependent to driver parameter)

            RANGE_MAX –
                 defines upper limit of the allowed value range for this
                 parameter, can be an integer value as well as a dotted
                 number, this parameter is usually used in conjuncion
                 with 'RANGE_MIN' but may alo appear without
                 (optional returned, dependent to driver parameter)

            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_DRIVER_PARAMETER INFO ALSA ACTIVE"
        S: "DESCRIPTION: Wether device is enabled"
           "TYPE: BOOL"
           "MANDATORY: false"
           "FIX: false"
           "MULTIPLICITY: false"
           "DEFAULT: TRUE"
           "."

4.3.4 Creating a MIDI input device

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

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

Where <midi-input-driver> 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[<device-id>]" -
        in case the device was successfully created, where
        <device-id> is the numerical ID of the new device

    "WRN[<device-id>]:<warningcode>:<warningmessage>" -
        in case the driver was loaded successfully, where
        <device-id> is the numerical ID of the new device, 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[0]"

4.3.5 Destroying a MIDI input device

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

    DESTROY MIDI_INPUT_DEVICE <device-id>

Where <device-id> should be replaced by the device's numerical ID.

Possible Answers:

    "OK" -
        in case the device was successfully destroyed

    "WRN:<warningcode>:<warningmessage>" -
        in case the device was destroyed, 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 0"
S: "OK"

4.3.6 Getting all MIDI input devices

Use the following command to list all created MIDI input devices:

GET MIDI_INPUT_DEVICES

Possible Answers:

LinuxSampler will answer by sending a comma separated list
with the numerical Ids of all created MIDI input devices.

Examples:

C: "GET MIDI_INPUT_DEVICES"
S: "0,1,2"

C: "GET MIDI_INPUT_DEVICES"
S: "1,3"

4.3.7 Getting current settings of a MIDI input device

Use the following command to get current settings of a specific,
created MIDI input device:

GET MIDI_INPUT_DEVICE INFO <device-id>

Where <device-id> is the numerical ID of the MIDI input device.

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):

        DRIVER –
            identifier of the used MIDI input driver, as e.g.
            returned by the "GET AVAILABLE_MIDI_INPUT_DRIVERS"
            command

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

    The mentioned fields above don't have to be in particular
    order. The fields above are only those fields which are
    returned by all MIDI input devices. Every MIDI input driver
    might have its own, additional driver specific parameters (see
    "GET MIDI_INPUT_DRIVER INFO" command) which are also returned
    by this command.

    Example:

    C: "GET MIDI_INPUT_DEVICE INFO 0"
    S: "DRIVER: ALSA"
       "ACTIVE: true"
       "."

4.3.8 Changing settings of audio output devices

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

        SET MIDI_INPUT_DEVICE_PARAMETER <device-id> <key> <value>

    or alternatively:

        SET MIDI_INPUT_DEVICE_PARAMETER <device-id> <key>=<value>

    Where <device-id> should be replaced by the numerical ID of the
    MIDI input device, <key> by the name of the parameter to change 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 0 ACTIVE false"
S: "OK"

C: "SET MIDI_INPUT_DEVICE PARAMETER 0 ACTIVE=false"
S: "OK"

4.3.9 Getting informations about a MIDI port

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

GET MIDI_INPUT_PORT INFO <device-id> <midi-port>

Where <device-id> is the numerical ID of the MIDI inpupt device 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 0 0"
S: "NAME: Masterkeyboard"
   "ALSA_SEQ_BINDINGS: '64:0'"
   "."

4.3.10 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 <dev-id> <port> <param>

Where <dev-id> is the numerical ID of the MIDI input device as
returned by the "GET MIDI_INPUT_DEVICES" 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:

        TYPE –
            either "BOOL" for boolean value(s) or "INT" for integer
            value(s) or "FLOAT" for dotted number(s) or "STRING" for
            character string(s)
            (always returned)

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

        FIX –
            either true or false, if true then this parameter is
            read only, thus 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, this parameter is usually used in conjunction

with 'RANGE_MAX' but may alo appear without
(optionally returned, dependent to driver & port
parameter)

RANGE_MAX –
defines upper limit of the allowed value range for this
parameter, can be an integer value as well as a dotted
number, this parameter is usually used in conjunction
with 'RANGE_MIN' but may alo appear without
(optionally returned, dependent to driver & port
parameter)

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

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

Example:

C: "GET MIDI_INPUT_PORT_PARAMETER INFO 0 0 ALSA_SEQ_BINDINGS"
S: "DESCRIPTION: bindings to other ALSA sequencer clients"
   "TYPE: STRING"
   "FIX: false"
   "MULTIPLICITY: true"
   "POSSIBILITES: '64:0','68:0','68:1'"
   "."

4.3.11 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 <device-id> <port> <key> <value>

or alternatively:

   SET MIDI_INPUT_PORT PARAMETER <device-id> <port> <key>=<value>

Where <device-id> should be replaced by the numerical ID of the
MIDI device, <port> by the MIDI port number, <key> by the name of
the parameter to change 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 0 0 ALSA_SEQ_BINDINGS 64:0"
S: "OK"

C: "SET MIDI_INPUT_PORT PARAMETER 0 0 ALSA_SEQ_BINDINGS=64:0"
S: "OK"

C: "SET MIDI_INPUT_PORT PARAMETER 0 0 NAME=My Masterkeyboard"
S: "OK"

4.4   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.

4.4.1 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

4.4.2 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

4.4.3 Getting all created 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 by returning a comma separated list
> with all sampler channels.

Example:

> C: "GET CHANNELS"
> S: "0,1,2,3,4,5,6,9,10,11,15,20"

4.4.4 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

4.4.5 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

4.4.6 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"

4.4.7 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"
       "."

4.4.8 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_DEVICE –
    numerical ID of the audio output device which is
    currently connected to this sampler channel to output
    the audio signal, "NONE" if there's no device
    connected to this sampler channel

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_DEVICE –
    numerical ID of the MIDI input device which is
    currently connected to this sampler channel to deliver
    MIDI input commands, "NONE" if there's no device
    connected to this sampler channel

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_DEVICE: 0"
           "AUDIO_OUTPUT_CHANNELS: 2"
           "AUDIO_OUTPUT_ROUTING: 0,1"
           "INSTRUMENT_FILE: /home/joe/FazioliPiano.gig"
           "INSTRUMENT_NR: 0"
           "MIDI_INPUT_DEVICE: 0"
           "MIDI_INPUT_PORT: 0"
           "MIDI_INPUT_CHANNEL: 5"
           "."

4.4.9 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.

4.4.10 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.

4.4.11 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: ""

4.4.12 Setting audio output device

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

SET CHANNEL AUDIO_OUTPUT_DEVICE <sampler-channel> <audio-device-id>

Where <audio-device-id> is the numerical ID of the audio output

device and <sampler-channel> is the respective sampler channel
number.

Possible Answers:

"OK" -
    on success

"WRN:<warningcode>:<warningmessage>" -
    if audio output device 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

4.4.13 Setting audio output type

DEPRECATED: THIS COMMAND WILL DISAPPEAR SOON!

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

4.4.14 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

4.4.15 Setting MIDI input device

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

    SET CHANNEL MIDI_INPUT_DEVICE <sampler-channel> <midi-device-id>

    Where <midi-device-id> is the numerical ID of the MIDI input
    device and <sampler-channel> is the respective sampler channel
    number.

    Possible Answers:

        "OK" -
            on success

        "WRN:<warningcode>:<warningmessage>" -
            if MIDI input device 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

4.4.16 Setting MIDI input type

DEPRECATED: THIS COMMAND WILL DISAPPEAR SOON!

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

SET CHANNEL MIDI_INPUT_TYPE <sampler-channel> <midi-input-type>

Where <midi-input-type> is currently only "ALSA" and
<sampler-channel> is the respective sampler channel number.

Possible Answers:

    "OK" -
        on success

    "WRN:<warningcode>:<warningmessage>" -
        if MIDI input 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

4.4.17 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

4.4.18 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

4.4.19 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

4.4.20 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

4.5   Controlling connection

The following commands are used to control the connection to
LinuxSampler.

4.5.1 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

4.5.2 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

4.5.3 Enable or disable echo of commands

To enable or disable back sending of commands to the client the
following command can be used:

    SET ECHO <value>

Where <value> should be replaced either by "1" to enable echo mode
or "0" to disable echo mode. When echo mode is enabled, all
commands send to LinuxSampler will be immediately send back and
after this echo the actual response to the command will be
returned. Echo mode will only be altered for the client connection
that issued the "SET ECHO" command, not globally for all client
connections.

Possible Answers:

    "OK" - always returned

4.5.4 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.

5. Command Syntax

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

 ADD <SP> CHANNEL

 CREATE <create-instruction>

 DESTROY <destroy-instruction>

 GET <SP> <get-instruction>

 LOAD <SP> <load-instruction>

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

    SET <SP> <set-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.

   <create-instruction> ::=
       AUDIO_OUTPUT_DEVICE <SP> <audio-output-driver> <SP> <param-list> |
       MIDI_INPUT_DEVICE <SP> <midi-input-driver> <SP> <param-list>

   <destroy-instruction> ::=
        AUDIO_OUTPUT_DEVICE <SP> <device-id> |
        MIDI_INPUT_DEVICE <SP> <device-id>

   <set-instruction> ::=
                CHANNEL <SP> <set-chan-instruction> |
                AUDIO_OUTPUT_DEVICE_PARAMETER <SP> <device-id> <SP>
                    <key> <SP> <value> |
                AUDIO_OUTPUT_DEVICE_PARAMETER <SP> <device-id> <SP>
                    <key> = <value>
                AUDIO_OUTPUT_CHANNEL_PARAMETER <SP> <device-id> <SP>
                    <audio-chan> <SP> <key> <SP> <value> |
                AUDIO_OUTPUT_CHANNEL_PARAMETER <SP> <device-id> <SP>
                    <audio-chan> <SP> <key> = <value> |
                MIDI_INPUT_DEVICE_PARAMETER <SP> <device-id> <SP> <key>
                    <SP> <value> |
                MIDI_INPUT_DEVICE_PARAMETER <SP> <device-id> <SP> <key>
                    = <value> |
                MIDI_INPUT_PORT <SP> PARAMETER <SP> <device-id> <SP>
                    <port> <SP> <key> <SP> <value> |
                MIDI_INPUT_PORT <SP> PARAMETER <SP> <device-id> <SP>
                    <port> <SP> <key> = <value>
                ECHO <SP> <bool>

   <get-instruction> ::=
                AUDIO_OUTPUT_DEVICES |
                AUDIO_OUTPUT_DEVICE <SP> INFO <SP> <device-id> |
                AUDIO_OUTPUT_CHANNEL <SP> INFO <SP> <device-id> <SP>

```
                  <audio-chan> |
              AUDIO_OUTPUT_CHANNEL_PARAMETER <SP> INFO <SP>
                  <device-id> <SP> <audio-chan> <SP> <parameter> |
              AVAILABLE_AUDIO_OUTPUT_DRIVERS |
              AVAILABLE_MIDI_INPUT_DRIVERS |
              MIDI_INPUT_DEVICES |
              MIDI_INPUT_DEVICE <SP> INFO <SP> <device-id> |
              MIDI_INPUT_DRIVER <SP> INFO <SP> <midi-input-driver> |
              MIDI_INPUT_DRIVER_PARAMETER <SP> INFO <SP>
                  <midi-input-driver> <SP> <param> <SP>
                  <dependency-list> |
              MIDI_INPUT_PORT <SP> INFO <SP> <device-id> <SP>
                  <midi-port> |
              MIDI_INPUT_PORT_PARAMETER <SP> INFO <SP> <device-id>
                  <SP> <port> <SP> <param> |
              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_DEVICE <SP> <sampler-channel> <SP>
                  <audio-device-id> |
              AUDIO_OUTPUT_CHANNEL <SP> <sampler-channel> <SP>
                  <audio-output-channel> <SP> <audio-output-channel> |
              AUDIO_OUTPUT_TYPE <SP> <sampler-channel> <SP>
                  <audio-output-driver> |
              MIDI_INPUT_DEVICE <SP> <sampler-channel> <SP>
                  <midi-device-id> |
              MIDI_INPUT_TYPE <SP> <sampler-channel> <SP>
                  <mid-input-driver> |
              MIDI_INPUT_PORT <SP> <sampler-channel> <SP>
                  <midi-input-port> |
              MIDI_INPUT_CHANNEL <SP> <sampler-channel> <SP>
                  <midi-input-channel> |
              VOLUME <SP> <sampler-channel> <SP> <volume>

    <device-id> ::= <number>
```

```
   <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-driver> ::= 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 | ALL

   <midi-input-type> ::= ALSA

   <volume> ::= <dotnum>

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

   <filename> ::= <string>

   <dependency-list> ::= <parameter-list>

   <parameter-list> ::= <epsilon> |
                        <string> = ' <string> ' |
                        <string> = <number> |
                        <string> = <dotnum> |
                        <string> = <bool> |
                        <string> = ' <string> ' <SP> <dependency-list>
                        <string> = <number> <SP> <dependency-list>
                        <string> = <dotnum> <SP> <dependency-list>
                        <string> = <bool> <SP> <dependency-list>

   <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

<bool> ::= TRUE | true | 1 | FALSE | false | 0

<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.

6. Events and special UDP packets

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

6.1 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.

6.2 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.

6.3 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.

6.4 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.

6.5 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.

6.6   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.

6.7   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)

7.   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.

Acknowledgments

Author's Addresses

Christian Schoenebeck
Interessengemeinschaft Software Engineering e. V.
Max-Planck-Str. 39
74081 Heilbronn
Germany
Email: schoenebeck at software minus engineering dot org

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