LinuxSampler Developers	C. Schoenebeck
Internet-Draft	LinuxSampler.org
Intended status: Standards Track	July 15, 2015
Expires: January 16, 2016	

LinuxSampler Control Protocol
LSCP 1.7

Abstract

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

Status of this Memo

Table of Contents

TOC

1. Requirements notation

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 [RFC2119].

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

In examples, "C:" and "S:" indicate lines sent by the client (front-end) 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 [RFC20]), 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 message "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.

---

2. Versioning of this specification

LSCP will certainly be extended and enhanced by-and-by. Each official release of the LSCP specification will be tagged with a unique version tuple. The version tuple consists at least of a major and minor version number like:

    "1.2"

In this example the major version number would be "1" and the minor version number would be "2". Note that the version tuple might also have more than two elements. The major version number defines a group of backward compatible versions. That means a frontend is compatible to the connected sampler if and only if the LSCP versions to which each of the two parties complies to, match both of the following rules:

Compatibility:

1. The frontend's LSCP major version and the sampler's LSCP major version are exactly equal.
2. The frontend's LSCP minor version is less or equal than the sampler's LSCP minor version.

Compatibility can only be claimed if both rules are true. The frontend can use the **"GET SERVER INFO"** command to get the version of the LSCP specification the sampler complies with.

---

3. Introduction

LinuxSampler is a so called software sampler application capable to playback audio samples from a computer's Random Access 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 channel will be associated with an instance 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.

---

4. Focus of this protocol

Main focus of this protocol is to provide a way to configure a running LinuxSampler instance and to retrieve information about it. 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.

5. Communication Overview

TOC

There are two distinct methods of communication between a running instance of LinuxSampler and one or more control applications, so called "front-ends": a simple request/response communication method used by the clients to give commands to the server as well as to inquire about server's status and a subscribe/notify communication method used by the client to subscribe to and receive notifications of certain events as they happen on the server. The latter needs more effort to be implemented in the front-end application. The two communication methods will be described next.

5.1. Request/response communication method

TOC

This simple communication method is based on **TCP** [RFC793]. The front-end application establishes a TCP connection to the LinuxSampler instance on a certain host system. Then the front-end 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 front-ends, 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 command, instead it should keep the connection established and simply resend message(s) for subsequent commands. To keep information in the front-end up-to-date the front-end has to periodically send new requests to get the current information from the LinuxSampler instance. This is often referred to as "polling". While polling is simple to implement and may be OK to use in some cases, there may be disadvantages to polling such as network traffic overhead and information being out of date. It is possible for a client or several clients to open more than one connection to the server at the same time. It is also possible to send more than one request to the server at the same time but if those requests are sent over the same connection server MUST execute them sequentially. Upon executing a request server will produce a result set and send it to the client. Each and every request made by the client MUST result in a result set being sent back to the client. No other data other than a result set may be sent by a server to a client. No result set may be sent to a client without the client sending request to the server first. On any particular connection, result sets MUST be sent in their entirety without being interrupted by other result sets. If several requests got queued up at the server they MUST be processed in the order they were received and result sets MUST be sent back in the same order.

5.1.1. Result format

TOC

Result set could be one of the following types:

1. Normal
2. Warning
3. Error

Warning and Error result sets MUST be single line and have the following format:

- "WRN:<warning-code>:<warning-message>"

- "ERR:<error-code>:<error-message>"

Where <warning-code> and <error-code> are numeric unique identifiers of the warning or error and <warning-message> and <error-message> are human readable descriptions of the warning or error respectively.

Examples:

C: "LOAD INSTRUMENT '/home/me/Boesendorfer24bit.gig" 0 0

S: "WRN:32:This is a 24 bit patch which is not supported natively yet."

C: "GET AUDIO_OUTPUT_DRIVER_PARAMETER INFO ALSA EAR"

S: "ERR:3456:Audio output driver 'ALSA' does not have a parameter 'EAR'."

C: "GET AUDIO_OUTPUT_DEVICE INFO 123456"

S: "ERR:9:There is no audio output device with index 123456."

Normal result sets could be:

1. Empty
2. Single line
3. Multi-line

Empty result set is issued when the server only needed to acknowledge the fact that the request was received and it was processed successfully and no additional information is available. This result set has the following format:

"OK"

Example:

C: "SET AUDIO_OUTPUT_DEVICE_PARAMETER 0 CHANNELS=4"

S: "OK"

Single line result sets are command specific. One example of a single line result set is an empty line. Multi-line result sets are command specific and may include one or more lines of information. They MUST always end with the following line:

"."

Example:

C: "GET AUDIO_OUTPUT_DEVICE INFO 0"

S: "DRIVER: ALSA"

"CHANNELS: 2"

"SAMPLERATE: 44100"

"ACTIVE: true"

"FRAGMENTS: 2"

"FRAGMENTSIZE: 128"

"CARD: '0,0'"

"."

In addition to above mentioned formats, warnings and empty result sets MAY be indexed. In this case, they have the following formats respectively:

- "WRN[<index>]:<warning-code>:<warning-message>"
- "OK[<index>]"

where <index> is command specific and is used to indicate channel number that the result set was related to or other integer value.

Each line of the result set MUST end with <CRLF>.

Examples:

C: "ADD CHANNEL"

S: "OK[12]"

C: "CREATE AUDIO_OUTPUT_DEVICE ALSA SAMPLERATE=96000"

S: "WRN[0]:32:Sample rate not supported, using 44100 instead."

5.2. Subscribe/notify communication method

This more sophisticated communication method is actually only an extension of the simple request/response communication method. The front-end still uses a TCP connection and sends the same commands on the TCP connection. Two extra commands are SUBSCRIBE and UNSUBSCRIBE commands that allow a client to tell the server that it is interested in receiving notifications about certain events as they happen on the server. The SUBSCRIBE command has the following syntax:

SUBSCRIBE <event-id>

where <event-id> will be replaced by the respective event that client wants to subscribe to. Upon receiving such request, server SHOULD respond with OK and start sending EVENT notifications when a given even has occurred to the front-end when an event has occurred. It MAY be possible certain events may be sent before OK response during real time nature of their generation. Event messages have the following format:

NOTIFY:<event-id>:<custom-event-data>

where <event-id> uniquely identifies the event that has occurred and <custom-event-data> is event specific.

Several rules must be followed by the server when generating events:

1. Events MUST NOT be sent to any client who has not issued an appropriate SUBSCRIBE command.
2. Events MUST only be sent using the same connection that was used to subscribe to them.
3. When response is being sent to the client, event MUST be inserted in the stream before or after the response, but NOT in the middle. Same is true about the response. It should never be inserted in the middle of the event message as well as any other response.

If the client is not interested in a particular event anymore it MAY issue UNSUBSCRIBE command using the following syntax:

```
UNSUBSCRIBE <event-id>
```

where <event-id> will be replace by the respective event that client is no longer interested in receiving. For a list of supported events see **Section 8**.

Example: the fill states of disk stream buffers have changed on sampler channel 4 and the LinuxSampler instance will react by sending the following message to all clients who subscribed to this event:

```
NOTIFY:CHANNEL_BUFFER_FILL:4 [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%.

Clients may choose to open more than one connection to the server and use some connections to receive notifications while using other connections to issue commands to the back-end. This is entirely legal and up to the implementation. This does not change the protocol in any way and no special restrictions exist on the server to allow or disallow this or to track what connections belong to what front-ends. Server will listen on a single port, accept multiple connections and support protocol described in this specification in it's entirety on this single port on each connection that it accepted.

Due to the fact that TCP is used for this communication, dead peers will be detected automatically by the OS TCP stack. While it may take a while to detect dead peers if no traffic is being sent from server to client (TCP keep-alive timer is set to 2 hours on many OSes) it will not be an issue here as when notifications are sent by the server, dead client will be detected quickly.

When connection is closed for any reason server MUST forget all subscriptions that were made on this connection. If client reconnects it MUST resubscribe to all events that it wants to receive.

TOC

6. 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.

TOC

6.1. 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.

TOC

6.2. Configuring audio drivers

Instances of drivers in LinuxSampler are called devices. You can use multiple audio devices simultaneously, 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 front-ends can be written independently of what drivers are currently implemented and what parameters these drivers are actually offering. This means front-ends can even handle drivers which are implemented somewhere in future without modifying the front-end 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.

6.2.1. Getting amount of available audio output drivers

TOC

Use the following command to get the number of audio output drivers currently available for the LinuxSampler instance:

```
GET AVAILABLE_AUDIO_OUTPUT_DRIVERS
```

Possible Answers:

LinuxSampler will answer by sending the number of audio output drivers.

Example:

```
C: "GET AVAILABLE_AUDIO_OUTPUT_DRIVERS"

S: "2"
```

6.2.2. Getting all available audio output drivers

TOC

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

```
LIST AVAILABLE_AUDIO_OUTPUT_DRIVERS
```

Possible Answers:

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

Example:

```
C: "LIST AVAILABLE_AUDIO_OUTPUT_DRIVERS"

S: "ALSA,JACK"
```

6.2.3. Getting information about a specific audio output driver

TOC

Use the following command to get detailed information 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 **"LIST 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"

   "."
```

6.2.4. Getting information about specific audio output driver parameter

Use the following command to get detailed information 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 **"LIST 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 front-end 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 information which is always returned, independently of the given driver parameter and there are optional information which is 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 parameters 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)

POSSIBILITIES -

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'"

"POSSIBILITIES: '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"

"."

6.2.5. Creating an audio output device

TOC

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 as returned by the **"LIST AVAILABLE_AUDIO_OUTPUT_DRIVERS"** command 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 this information.

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>]:<warning-code>:<warning-message>" -

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 fall-back values), providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

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]"

6.2.6. 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 **"LIST AUDIO_OUTPUT_DEVICES"** command.

Possible Answers:

"OK" -

in case the device was successfully destroyed

"WRN:<warning-code>:<warning-message>" -

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:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

C: "DESTROY AUDIO_OUTPUT_DEVICE 0"

S: "OK"

6.2.7. Getting all created audio output device count

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

GET AUDIO_OUTPUT_DEVICES

Possible Answers:

LinuxSampler will answer by sending the current number of all audio output devices.

Example:

C: "GET AUDIO_OUTPUT_DEVICES"

S: "4"

6.2.8. Getting all created audio output device list

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

```
LIST AUDIO_OUTPUT_DEVICES
```

Possible Answers:

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

Example:

```
C: "LIST AUDIO_OUTPUT_DEVICES"

S: "0,1,4,5"
```

6.2.9. 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 replaced by numerical ID of the audio output device as e.g. returned by the **"LIST 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 **"LIST 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 **Section 6.2.3**) 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'"

"."

6.2.10. Changing settings of audio output devices

TOC

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

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

Where <device-id> should be replaced by the numerical ID of the audio output device as given by the **"CREATE AUDIO_OUTPUT_DEVICE"** or **"LIST AUDIO_OUTPUT_DEVICES"** command, <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:<warning-code>:<warning-message>" -

in case setting was changed successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

C: "SET AUDIO_OUTPUT_DEVICE_PARAMETER 0 FRAGMENTSIZE=128"

S: "OK"

6.2.11. Getting information about an audio channel

TOC

Use the following command to get information 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 as given by the **"CREATE AUDIO_OUTPUT_DEVICE"** or **"LIST AUDIO_OUTPUT_DEVICES"** command 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 mix-channel 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 -
> >
> > > numerical ID (positive integer including 0) which 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 and 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'"

"."

6.2.12. Getting information about specific audio channel parameter

Use the following command to get detailed information about 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 **"CREATE AUDIO_OUTPUT_DEVICE"** or **"LIST 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 information which is always returned, independently of the given channel parameter and there is optional information which is 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 and 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 and channel parameter)

POSSIBILITIES -

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

"POSSIBILITIES: 'PCM:0','PCM:1','ardour:0','ardour:1'"

"."

6.2.13. Changing settings of audio output channels

TOC

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

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

Where <dev-id> should be replaced by the numerical ID of the audio output device as returned by the **"CREATE AUDIO_OUTPUT_DEVICE"** or **"LIST AUDIO_OUTPUT_DEVICES"** command, <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:<warning-code>:<warning-message>" -

in case setting was changed successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

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"

TOC

6.3. Configuring MIDI input drivers

Instances of drivers in LinuxSampler are called devices. You can use multiple MIDI devices simultaneously, 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 front-ends can be written independently of what drivers are currently implemented and what parameters these drivers are actually offering. This means front-ends can even handle drivers which are implemented somewhere in future without modifying the front-end 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.

TOC

6.3.1. Getting amount of available MIDI input drivers

Use the following command to get the number of MIDI input drivers currently available for the LinuxSampler instance:

GET AVAILABLE_MIDI_INPUT_DRIVERS

Possible Answers:

LinuxSampler will answer by sending the number of available MIDI input drivers.

Example:

C: "GET AVAILABLE_MIDI_INPUT_DRIVERS"

S: "2"

TOC

6.3.2. Getting all available MIDI input drivers

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

LIST AVAILABLE_MIDI_INPUT_DRIVERS

Possible Answers:

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

Example:

C: "LIST AVAILABLE_MIDI_INPUT_DRIVERS"

S: "ALSA,JACK"

TOC

6.3.3. Getting information about a specific MIDI input driver

Use the following command to get detailed information 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 as returned by the **"LIST AVAILABLE_MIDI_INPUT_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 -

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"
   "."
```

6.3.4. Getting information about specific MIDI input driver parameter

TOC

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

```
GET MIDI_INPUT_DRIVER_PARAMETER INFO <midit> <param> [<deplist>]
```

Where <midit> is the name of the MIDI input driver as returned by the **"LIST 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 front-end 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 is information which is always returned, independent of the given driver parameter and there is optional information which is 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 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 MIDI_INPUT_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 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 single value allowed (always returned, no matter which driver parameter)

DEPENDS -

comma separated list of parameters 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 MIDI_INPUT_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)

POSSIBILITIES -

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.

Example:

```
C: "GET MIDI_INPUT_DRIVER_PARAMETER INFO ALSA ACTIVE"

S: "DESCRIPTION: Whether device is enabled"

  "TYPE: BOOL"

  "MANDATORY: false"

  "FIX: false"

  "MULTIPLICITY: false"
```

"DEFAULT: true"

"."

6.3.5. Creating a MIDI input device

TOC

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 as returned by the **"LIST AVAILABLE_MIDI_INPUT_DRIVERS"** command 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 that information.

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>]:<warning-code>:<warning-message>" -

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:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

C: "CREATE MIDI_INPUT_DEVICE ALSA"

S: "OK[0]"

6.3.6. Destroying a MIDI input device

TOC

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 as returned by the **"CREATE MIDI_INPUT_DEVICE"** or **"LIST MIDI_INPUT_DEVICES"** command.

Possible Answers:

"OK" -

in case the device was successfully destroyed

"WRN:<warning-code>:<warning-message>" -

in case the device was destroyed, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

C: "DESTROY MIDI_INPUT_DEVICE 0"

S: "OK"

TOC

6.3.7. Getting all created MIDI input device count

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

GET MIDI_INPUT_DEVICES

Possible Answers:

LinuxSampler will answer by sending the current number of all MIDI input devices.

Example:

C: "GET MIDI_INPUT_DEVICES"

S: "3"

TOC

6.3.8. Getting all created MIDI input device list

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

LIST 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: "LIST MIDI_INPUT_DEVICES"

S: "0,1,2"

C: "LIST MIDI_INPUT_DEVICES"

S: "1,3"

TOC

6.3.9. 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 as returned by the **"CREATE MIDI_INPUT_DEVICE"** or **"LIST MIDI_INPUT_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 (independent of driver):

DRIVER -

identifier of the used MIDI input driver, as e.g. returned by the **"LIST 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"

   "."
```

6.3.10. Changing settings of MIDI input devices

TOC

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

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

Where <device-id> should be replaced by the numerical ID of the MIDI input device as returned by the **"CREATE MIDI_INPUT_DEVICE"** or **"LIST MIDI_INPUT_DEVICES"** command, <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:<warning-code>:<warning-message>" -

in case setting was changed successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

C: "SET MIDI_INPUT_DEVICE_PARAMETER 0 ACTIVE=false"

S: "OK"

6.3.11. Getting information about a MIDI port

TOC

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

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

Where <device-id> is the numerical ID of the MIDI input device as returned by the **"CREATE MIDI_INPUT_DEVICE"** or **"LIST MIDI_INPUT_DEVICES"** command 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 and port. Every MIDI port might have its own, additional driver and port specific parameters.

Example:

C: "GET MIDI_INPUT_PORT INFO 0 0"

S: "NAME: 'Masterkeyboard'"

"ALSA_SEQ_BINDINGS: '64:0'"

"."

6.3.12. Getting information about specific MIDI port parameter

TOC

Use the following command to get detailed information about 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 **"CREATE MIDI_INPUT_DEVICE"** or **"LIST 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 is information which is always returned, independently of the given channel parameter and there is optional information 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 also appear without (optionally returned, dependent to driver and 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 also appear without (optionally returned, dependent to driver and port parameter)
>
> POSSIBILITIES -
>
>> comma separated list of possible values for this parameter, character strings are encapsulated into apostrophes (optionally returned, dependent to device and 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"

"POSSIBILITIES: '64:0','68:0','68:1'"

"."

6.3.13. 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>

Where <device-id> should be replaced by the numerical ID of the MIDI device as returned by the **"CREATE MIDI_INPUT_DEVICE"** or **"LIST MIDI_INPUT_DEVICES"** command, <port> by the MIDI port number, <key> by the name of the parameter to change and <value> by the new value for this parameter (encapsulated into apostrophes) or NONE (not encapsulated into apostrophes) for specifying no value for parameters allowing a list of values.

Possible Answers:

"OK" -

in case setting was successfully changed

"WRN:<warning-code>:<warning-message>" -

in case setting was changed successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

C: "SET MIDI_INPUT_PORT_PARAMETER 0 0 ALSA_SEQ_BINDINGS='20:0'"

S: "OK"

C: "SET MIDI_INPUT_PORT_PARAMETER 0 0 ALSA_SEQ_BINDINGS=NONE"

S: "OK"

6.4. Configuring sampler channels

The following commands describe how to add and remove sampler channels, associate a sampler channel with a sampler engine, load instruments and connect sampler channels to MIDI and audio devices.

6.4.1. Loading an instrument

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

```
LOAD INSTRUMENT [NON_MODAL] '<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.

Notice: since LSCP 1.2 the <filename> argument supports escape characters for special characters (see chapter "**Character Set and Escape Sequences**" for details) and accordingly backslash characters in the filename MUST now be escaped as well!

The difference between regular and NON_MODAL versions of the command is that the regular command returns OK only after the instrument has been fully loaded and the channel is ready to be used while NON_MODAL version returns immediately and a background process is launched to load the instrument on the channel. The **GET CHANNEL INFO** command can be used to obtain loading progress from INSTRUMENT_STATUS field. LOAD command will perform sanity checks such as making sure that the file could be read and it is of a proper format and SHOULD return ERR and SHOULD not launch the background process should any errors be detected at that point.

Possible Answers:

> "OK" -
>
> > in case the instrument was successfully loaded
>
> "WRN:<warning-code>:<warning-message>" -
>
> > 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:<error-code>:<error-message>" -
>
> > in case it failed, providing an appropriate error code and error message

Example (Unix):

```
C: LOAD INSTRUMENT '/home/joe/gigs/cello.gig' 0 0

S: OK
```

Example (Windows):

```
C: LOAD INSTRUMENT 'D:/MySounds/cello.gig' 0 0

S: OK
```

TOC

6.4.2. Loading a sampler engine

A sampler engine type can be associated to a specific sampler channel by the following command:

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

Where <engine-name> is an engine name as obtained by the **"LIST AVAILABLE_ENGINES"** command and <sampler-channel> the sampler channel as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command where the engine type should be assigned to. This command should be issued after adding a new sampler channel and before any other control commands on the new sampler channel. It can also be used to change the engine type of a sampler channel. This command has (currently) no way to define or force if a new engine instance should be created and assigned to the given sampler channel or if an already existing instance of that engine type, shared with other sampler channels, should be used.

Possible Answers:

"OK" -

in case the engine was successfully deployed

"WRN:<warning-code>:<warning-message>" -

in case the engine was deployed successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

6.4.3. Getting all created sampler channel count

TOC

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

```
GET CHANNELS
```

Possible Answers:

LinuxSampler will answer by returning the current number of sampler channels.

Example:

```
C: "GET CHANNELS"

S: "12"
```

6.4.4. Getting all created sampler channel list

TOC

The number of sampler channels can change on runtime. To get the current list of sampler channels, the front-end can send the following command:

```
LIST CHANNELS
```

Possible Answers:

LinuxSampler will answer by returning a comma separated list with all sampler channels numerical IDs.

Example:

C: "LIST CHANNELS"

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

6.4.5. Adding a new sampler channel

TOC

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 front-end should send the respective, related commands right after to e.g. load an engine, load an instrument and setting input, output method and eventually other commands to initialize the new channel. The front-end 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 front-ends 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 be used to set up the sampler channel by sending subsequent initialization commands

"WRN:<warning-code>:<warning-message>" -

in case a new channel was added successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

6.4.6. Removing a sampler channel

TOC

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

REMOVE CHANNEL <sampler-channel>

Where <sampler-channel> should be replaced by the number of the sampler channel as given by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command. The channel numbers of all subsequent sampler channels remain the same.

Possible Answers:

"OK" -

in case the given sampler channel could be removed

"WRN:<warning-code>:<warning-message>" -

in case the given channel was removed, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

6.4.7. Getting amount of available engines

TOC

The front-end can ask for the number of available engines by sending the following command:

GET AVAILABLE_ENGINES

Possible Answers:

LinuxSampler will answer by sending the number of available engines.

Example:

C: "GET AVAILABLE_ENGINES"

S: "4"

6.4.8. Getting all available engines

TOC

The front-end can ask for a list of all available engines by sending the following command:

LIST AVAILABLE_ENGINES

Possible Answers:

LinuxSampler will answer by sending a comma separated list of the engines' names encapsulated into apostrophes ('). Engine names can consist of lower and upper cases, digits and underlines ("_" character).

Example:

C: "LIST AVAILABLE_ENGINES"

S: "'gig','sfz','sf2'"

6.4.9. Getting information about an engine

TOC

The front-end can ask for information about a specific engine by sending the following command:

GET ENGINE INFO <engine-name>

Where <engine-name> is an engine name as obtained by the **"LIST AVAILABLE_ENGINES"** 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 categories are defined:
>
> DESCRIPTION -
>
> > arbitrary description text about the engine (note that the character string may contain **escape sequences**)
>
> VERSION -
>
> > arbitrary character string regarding the engine's version

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

Examples:

> C: "GET ENGINE INFO gig"
>
> S: "DESCRIPTION: GigaSampler Format Engine"
>
> "VERSION: 1.110"
>
> "."
>
> C: "GET ENGINE INFO sf2"
>
> S: "DESCRIPTION: SoundFont Format Engine"
>
> "VERSION: 1.4"
>
> "."
>
> C: "GET ENGINE INFO sfz"
>
> S: "DESCRIPTION: SFZ Format Engine"
>
> "VERSION: 1.11"
>
> "."

6.4.10. Getting sampler channel information

TOC

The front-end 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 front-end is interested in as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command.

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 associated with the sampler channel, "NONE" if there's no engine associated 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, "-1" 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 engine'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 (note: since LSCP 1.2 this path may contain **escape sequences**)

INSTRUMENT_NR -

> the instrument index number of the loaded instrument, "-1" if there's no instrument loaded for this sampler channel

INSTRUMENT_NAME -

> the instrument name of the loaded instrument (note: since LSCP 1.2 this character string may contain **escape sequences**)

INSTRUMENT_STATUS -

> Integer values 0 to 100 indicating loading progress percentage for the instrument. Negative value indicates a loading exception (also returns "-1" in case no instrument was yet to be loaded on the sampler channel). Value of 100 indicates that the instrument is fully loaded.

MIDI_INPUT_DEVICE -

> DEPRECATED: THIS FIELD WILL DISAPPEAR!
>
> numerical ID of the MIDI input device which is currently

connected to this sampler channel to deliver MIDI input commands, "-1" if there's no device connected to this sampler channel

Should not be used anymore as of LSCP v1.6 and younger. This field is currently only preserved for backward compatibility.

This field a relict from times where only one MIDI input per sampler channel was allowed. Use **"GET CHANNEL MIDI_INPUTS"** instead.

MIDI_INPUT_PORT -

DEPRECATED: THIS FIELD WILL DISAPPEAR!

port number of the MIDI input device (in case a MIDI device was already assigned to the sampler channel)

Should not be used anymore as of LSCP v1.6 and younger. This field is currently only preserved for backward compatibility.

This field a relict from times where only one MIDI input per sampler channel was allowed. Use **"GET CHANNEL MIDI_INPUTS"** instead.

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)

MUTE -

Determines whether the channel is muted, "true" if the channel is muted, "false" if the channel is not muted, and "MUTED_BY_SOLO" if the channel is muted because of the presence of a solo channel and will be unmuted when there are no solo channels left

SOLO -

Determines whether this is a solo channel, "true" if the channel is a solo channel; "false" otherwise

MIDI_INSTRUMENT_MAP -

Determines to which MIDI instrument map this sampler channel is assigned to. Read chapter **"SET CHANNEL MIDI_INSTRUMENT_MAP"** for a list of possible values.

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

Example:

```
C: "GET CHANNEL INFO 34"

S: "ENGINE_NAME: gig"

   "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"

   "INSTRUMENT_NAME: Fazioli Piano"

   "INSTRUMENT_STATUS: 100"

   "MIDI_INPUT_DEVICE: 0"

   "MIDI_INPUT_PORT: 0"

   "MIDI_INPUT_CHANNEL: 5"

   "VOLUME: 1.0"

   "MUTE: false"

   "SOLO: false"

   "MIDI_INSTRUMENT_MAP: NONE"

   "."
```

6.4.11. Current number of active voices

TOC

The front-end 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 front-end is interested in as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command.

Possible Answers:

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

Example:

6.4.12. Current number of active disk streams

TOC

The front-end 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 front-end is interested in as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command.

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.

Example:

6.4.13. Current fill state of disk stream buffers

TOC

The front-end 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 front-end is interested in as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command.

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 front-end has to sort them by itself if necessary.

Examples:

```
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: ""
```

6.4.14. Setting audio output device

TOC

The front-end 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 <sampler-channel> is the respective sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command and <audio-device-id> is the numerical ID of the audio output device as given by the **"CREATE AUDIO_OUTPUT_DEVICE"** or **"LIST AUDIO_OUTPUT_DEVICES"** command.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if audio output device was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

6.4.15. Setting audio output type

TOC

DEPRECATED: THIS COMMAND WILL DISAPPEAR!

The front-end 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:<warning-code>:<warning-message>" -

if audio output type was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

Deprecated:

Should not be used anymore. This command is currently only preserved for backward compatibility.

This command is a relict from times where there was no sophisticated driver management yet. Use **"CREATE AUDIO_OUTPUT_DEVICE"** and **"SET CHANNEL AUDIO_OUTPUT_DEVICE"** instead.

TOC

6.4.16. Setting audio output channel

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

```
SET CHANNEL AUDIO_OUTPUT_CHANNEL <sampler-chan> <audio-out> <audio-in>
```

Where <sampler-chan> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command, <audio-out> is the numerical ID of the sampler channel's audio output channel which should be rerouted and <audio-in> is the numerical ID of the audio channel of the selected audio output device where <audio-out> should be routed to.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if audio output channel was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

TOC

6.4.17. Add MIDI input to sampler channel

The front-end can add a MIDI input on a specific sampler channel by sending the following command:

```
ADD CHANNEL MIDI_INPUT <sampler-channel> <midi-device-id> [<midi-input-port>]
```

Where <sampler-channel> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command and <midi-device-id> is the numerical ID of the MIDI input device as returned by the **"CREATE MIDI_INPUT_DEVICE"** or **"LIST MIDI_INPUT_DEVICES"** command, and <midi-input-port> is an optional MIDI input port number of that MIDI input device. If <midi-input-port> is omitted, then the MIDI input device's first port (port number 0) is used.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if MIDI input port was connected, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

C: "ADD CHANNEL MIDI_INPUT 0 0"

S: "OK"

C: "ADD CHANNEL MIDI_INPUT 1 0"

S: "OK"

C: "ADD CHANNEL MIDI_INPUT 1 1 1"

S: "OK"

C: "ADD CHANNEL MIDI_INPUT 1 2 0"

S: "OK"

Since:

Introduced with LSCP v1.6

6.4.18. Remove MIDI input(s) from sampler channel

TOC

The front-end can remove one ore more MIDI input(s) on a specific sampler channel by sending the following command:

REMOVE CHANNEL MIDI_INPUT <sampler-channel> [<midi-device-id> [<midi-input-port>]]

Where <sampler-channel> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command and <midi-device-id> and <midi-input-port> are optional numerical IDs defining the MIDI input device and one of its MIDI ports as returned by the **"LIST CHANNEL MIDI_INPUTS"** command.

If <midi-input-port> is omitted, then all MIDI input ports of <midi-device-id> are disconnected from this sampler channel.

If both, <midi-device-id> and <midi-input-port> are omitted, then all MIDI input ports currently connected to this sampler channel are disconnected from this sampler channel.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if MIDI input porst were disconnected, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

C: "REMOVE CHANNEL MIDI_INPUT 0"

S: "OK"

C: "REMOVE CHANNEL MIDI_INPUT 1"

S: "OK"

C: "REMOVE CHANNEL MIDI_INPUT 1 2 0"

S: "OK"

Since:

Introduced with LSCP v1.6

6.4.19. Getting all MIDI inputs of a sampler channel

TOC

The front-end can query a list of all currently connected MIDI inputs of a certain sampler channel by sending the following command:

LIST CHANNEL MIDI_INPUTS <sampler-channel>

Where <sampler-channel> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command.

Possible Answers:

The sampler will answer by sending a comma separated list of MIDI input device ID - MIDI input port number pairs, where each pair is encapsulated into curly braces. The list is returned in one single line. The MIDI input device ID corresponds to the number returned by **"LIST MIDI_INPUT_DEVICES"** and the port number is the index of the respective MIDI port of that MIDI input device.

Example:

C: "LIST CHANNEL MIDI_INPUTS 0"

S: "{0,0},{1,3},{2,0}"

Since:

Introduced with LSCP v1.6

6.4.20. Setting MIDI input device

TOC

DEPRECATED: THIS COMMAND WILL DISAPPEAR!

The front-end 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 <sampler-channel> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command and <midi-device-id> is the numerical ID of the MIDI input device as returned by the **"CREATE MIDI_INPUT_DEVICE"** or **"LIST MIDI_INPUT_DEVICES"** command.

If more than 1 MIDI inputs are currently connected to this sampler channel: Sending this command will disconnect ALL currently connected MIDI input ports connected to this sampler channel before establishing the new MIDI input connection. So this command does NOT add the connection, it replaces all existing ones instead. This behavior is due to preserving full behavior backward compatibility.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if MIDI input device was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

Deprecated:

Should not be used anymore as of LSCP v1.6 and younger. This command is currently only preserved for backward compatibility.

This command is a relict from times where only one MIDI input per sampler channel was allowed. Use **"ADD CHANNEL MIDI_INPUT"** and **"REMOVE CHANNEL MIDI_INPUT"** instead.

TOC

6.4.21. Setting MIDI input type

DEPRECATED: THIS COMMAND WILL DISAPPEAR!

The front-end 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.

If more than 1 MIDI inputs are currently connected to this sampler channel: Sending this command will disconnect ALL currently connected MIDI input ports connected to this sampler channel before establishing the new MIDI input connection. So this command does NOT add the connection, it replaces all existing ones instead. This behavior is due to preserving full behavior backward compatibility.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if MIDI input type was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

Deprecated:

Should not be used anymore. This command is currently only preserved for backward compatibility.

This command is a relict from times where only 1 MIDI input per sampler channels was allowed and where no sophisticated driver management existed yet. Use **"ADD CHANNEL MIDI_INPUT"** and **"REMOVE CHANNEL MIDI_INPUT"** instead.

6.4.22. Setting MIDI input port

TOC

DEPRECATED: THIS COMMAND WILL DISAPPEAR!

The front-end can alter the MIDI input 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>.

If more than 1 MIDI inputs are currently connected to this sampler channel: Sending this command will switch the connection of the first (and only the first) MIDI input port currently being connected to this sampler channel, to another port of the same MIDI input device. Or in other words: the first MIDI input port currently connected to this sampler channel will be disconnected, and the requested other port of its MIDI input device will be connected to this sampler channel instead. This behavior is due to preserving full behavior backward compatibility.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if MIDI input port was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

Deprecated:

Should not be used anymore. This command is currently only preserved for backward compatibility.

This command is a relict from times where only one MIDI input per sampler channel was allowed. Use **"ADD CHANNEL MIDI_INPUT"** and **"REMOVE CHANNEL MIDI_INPUT"** instead.

6.4.23. Setting MIDI input channel

TOC

The front-end 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 number of the new MIDI input channel (zero indexed!) where <sampler-channel> should listen to, or "ALL" to listen on all 16 MIDI channels.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if MIDI input channel was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

C: "SET CHANNEL MIDI_INPUT_CHANNEL 0 0"

S: "OK"

C: "SET CHANNEL MIDI_INPUT_CHANNEL 1 ALL"

S: "OK"

6.4.24. Setting channel volume

TOC

The front-end 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 attenuation, 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:<warning-code>:<warning-message>" -

if channel volume was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

TOC

6.4.25. Muting a sampler channel

The front-end can mute/unmute a specific sampler channel by sending the following command:

```
SET CHANNEL MUTE <sampler-channel> <mute>
```

Where <sampler-channel> is the respective sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command and <mute> should be replaced either by "1" to mute the channel or "0" to unmute the channel.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if the channel was muted/unmuted, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

TOC

6.4.26. Soloing a sampler channel

The front-end can solo/unsolo a specific sampler channel by sending the following command:

```
SET CHANNEL SOLO <sampler-channel> <solo>
```

Where <sampler-channel> is the respective sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command and <solo> should be replaced either by "1" to solo the channel or "0" to unsolo the channel.

Possible Answers:

"OK" -

> on success

"WRN:<warning-code>:<warning-message>" -

> if the channel was soloed/unsoloed, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

> in case it failed, providing an appropriate error code and error message

Examples:

6.4.27. Assigning a MIDI instrument map to a sampler channel

The front-end can assign a MIDI instrument map to a specific sampler channel by sending the following command:

```
SET CHANNEL MIDI_INSTRUMENT_MAP <sampler-channel> <map>
```

Where <sampler-channel> is the respective sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command and <map> can have the following possibilites:

"NONE" -

> This is the default setting. In this case the sampler channel is not assigned any MIDI instrument map and thus will ignore all MIDI program change messages.

"DEFAULT" -

> The sampler channel will always use the default MIDI instrument map to handle MIDI program change messages.

numeric ID -

> You can assign a specific MIDI instrument map by replacing <map> with the respective numeric ID of the MIDI instrument map as returned by the **"LIST MIDI_INSTRUMENT_MAPS"** command. Once that map will be deleted, the sampler channel would fall back to "NONE".

Read chapter **"MIDI Instrument Mapping"** for details regarding MIDI instrument mapping.

Possible Answers:

"OK" -

> on success

"ERR:<error-code>:<error-message>" -

> in case it failed, providing an appropriate error code and error message

Examples:

6.4.28. Adding an effect send to a sampler channel

The front-end can create an additional effect send on a specific sampler channel by sending the following command:

```
CREATE FX_SEND <sampler-channel> <midi-ctrl> [<name>]
```

Where <sampler-channel> is the respective sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command, that is the sampler channel on which the effect send should be created on, <midi-ctrl> is a number between 0..127 defining the MIDI controller which can alter the effect send level and <name> is an optional argument defining a name for the effect send entity. The name does not have to be unique, but MUST be encapsulated into apostrophes and supports escape sequences as described in chapter "**Character Set and Escape Sequences**".

By default, that is as initial routing, the effect send's audio channels are automatically routed to the last audio channels of the sampler channel's audio output device, that way you can i.e. first increase the amount of audio channels on the audio output device for having dedicated effect send output channels and when "CREATE FX_SEND" is called, those channels will automatically be picked. You can alter the destination channels however with **"SET FX_SEND AUDIO_OUTPUT_CHANNEL"**.

Note: Create effect sends on a sampler channel only when needed, because having effect sends on a sampler channel will decrease runtime performance, because for implementing channel effect sends, separate (sampler channel local) audio buffers are needed to render and mix the voices and route the audio signal afterwards to the master outputs and effect send outputs (along with their respective effect send levels). A sampler channel without effect sends however can mix its voices directly into the audio output devices's audio buffers and is thus faster.

Possible Answers:

"OK[<fx-send-id>]" -

in case a new effect send could be added to the sampler channel, where <fx-send-id> reflects the unique ID of the newly created effect send entity

"ERR:<error-code>:<error-message>" -

when a new effect send could not be added, i.e. due to invalid parameters

Examples:

```
C: "CREATE FX_SEND 0 91 'Reverb Send'"

S: "OK[0]"

C: "CREATE FX_SEND 0 93"

S: "OK[1]"
```

6.4.29. Removing an effect send from a sampler channel

The front-end can remove an existing effect send on a specific sampler channel by sending the following command:

```
DESTROY FX_SEND <sampler-channel> <fx-send-id>
```

Where <sampler-channel> is the respective sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command, that is the sampler channel from which the effect send should be removed from and <fx-send-id> is the respective effect send number as returned by the **"CREATE FX_SEND"** or **"LIST FX_SENDS"** command.

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

C: "DESTROY FX_SEND 0 0"

S: "OK"

6.4.30. Getting amount of effect sends on a sampler channel

The front-end can ask for the amount of effect sends on a specific sampler channel by sending the following command:

```
GET FX_SENDS <sampler-channel>
```

Where <sampler-channel> is the respective sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command.

Possible Answers:

The sampler will answer by returning the number of effect sends on the given sampler channel.

Example:

C: "GET FX_SENDS 0"

S: "2"

6.4.31. Listing all effect sends on a sampler channel

The front-end can ask for a list of effect sends on a specific sampler channel by sending the following command:

```
LIST FX_SENDS <sampler-channel>
```

Where <sampler-channel> is the respective sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command.

Possible Answers:

The sampler will answer by returning a comma separated list with all effect sends' numerical IDs on the given sampler channel.

Examples:

C: "LIST FX_SENDS 0"

S: "0,1"

C: "LIST FX_SENDS 1"

S: ""

6.4.32. Getting effect send information

TOC

The front-end can ask for the current settings of an effect send entity by sending the following command:

GET FX_SEND INFO <sampler-channel> <fx-send-id>

Where <sampler-channel> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command and <fx-send-id> reflects the numerical ID of the effect send entity as returned by the **"CREATE FX_SEND"** or **"LIST FX_SENDS"** command.

Possible Answers:

The sampler 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:

NAME -

name of the effect send entity (note that this character string may contain **escape sequences**)

MIDI_CONTROLLER -

a value between 0 and 127 reflecting the MIDI controller which is able to modify the effect send's send level

LEVEL -

optionally dotted number reflecting the effect send's current send level (where a value < 1.0 means attenuation and a value > 1.0 means amplification)

AUDIO_OUTPUT_ROUTING -

comma separated list which reflects to which audio channel of the selected audio output device each effect send output channel is routed to, e.g. "0,3" would mean the effect send's output channel 0 is routed to channel 0 of the audio output device and the effect send's output channel 1 is routed to the channel 3 of the audio output device (see **"SET FX_SEND AUDIO_OUTPUT_CHANNEL"** for details), if an internal send effect is assigned to the effect send, then this setting defines the audio channel routing to that effect instance respectively

EFFECT -

destination send effect chain ID and destination effect chain position, separated by comma in the form "<effect-chain>, <chain-pos>" or "NONE" if there is no send effect assigned to the effect send

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

Example:

```
C: "GET FX_SEND INFO 0 0"

S: "NAME: Reverb Send"

  "MIDI_CONTROLLER: 91"

  "LEVEL: 0.3"

  "AUDIO_OUTPUT_ROUTING: 2,3"

  "EFFECT: NONE"

  "."

C: "GET FX_SEND INFO 0 1"

S: "NAME: Delay Send (Internal)"

  "MIDI_CONTROLLER: 93"

  "LEVEL: 0.51"

  "AUDIO_OUTPUT_ROUTING: 1,2"

  "EFFECT: 2,0"

  "."
```

TOC

6.4.33. Changing effect send's name

The front-end can alter the current name of an effect send entity by sending the following command:

```
SET FX_SEND NAME <sampler-chan> <fx-send-id> <name>
```

Where <sampler-chan> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command, <fx-send-id> reflects the numerical ID of the effect send entity as returned by the **"CREATE FX_SEND"** or **"LIST FX_SENDS"** command and <name> is the new name of the effect send entity, which does not have to be unique (name MUST be encapsulated into apostrophes and supports escape sequences as described in chapter "**Character Set and Escape Sequences**").

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

C: "SET FX_SEND NAME 0 0 'Fx Send 1'"

S: "OK"

6.4.34. Altering effect send's audio routing

TOC

The front-end can alter the destination of an effect send's audio channel on a specific sampler channel by sending the following command:

SET FX_SEND AUDIO_OUTPUT_CHANNEL <sampler-chan> <fx-send-id> <audio-src> <audio-dst>

Where <sampler-chan> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command, <fx-send-id> reflects the numerical ID of the effect send entity as returned by the **"CREATE FX_SEND"** or **"LIST FX_SENDS"** command, <audio-src> is the numerical ID of the effect send's audio channel which should be rerouted and <audio-dst> is the numerical ID of the audio channel of the selected audio output device where <audio-src> should be routed to. If an internal send effect is assigned to the effect send, then this setting defines the audio channel routing to that effect instance respectively.

Note that effect sends can only route audio to the same audio output device as assigned to the effect send's sampler channel. Also note that an effect send entity does always have exactly as much audio channels as its sampler channel. So if the sampler channel is stereo, the effect send does have two audio channels as well. Also keep in mind that the amount of audio channels on a sampler channel might be dependant not only to the deployed sampler engine on the sampler channel, but also dependant to the instrument currently loaded. However you can (effectively) turn an i.e. stereo effect send into a mono one by simply altering its audio routing appropriately.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if audio output channel was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

C: "SET FX_SEND AUDIO_OUTPUT_CHANNEL 0 0 0 2"

S: "OK"

6.4.35. Assigning destination effect to an effect send

TOC

The front-end can (re-)assign a destination effect to an effect send by sending the following command:

SET FX_SEND EFFECT <sampler-chan> <fx-send-id> <effect-chain> <chain-pos>

Where <sampler-chan> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command, <fx-send-id> reflects the numerical ID of the effect send entity as returned by the **"CREATE FX_SEND"** or **"LIST FX_SENDS"** command, <effect-chain> by the numerical ID of the destination effect chain as returned by the **"ADD SEND_EFFECT_CHAIN"** or **"LIST SEND_EFFECT_CHAINS"** command and <chain-pos> reflects the exact effect chain position in the effect chain which hosts the actual destination effect.

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

```
C: "SET FX_SEND EFFECT 0 0 2 5"

S: "OK"
```

6.4.36. Removing destination effect from an effect send

TOC

The front-end can (re-)assign a destination effect to an effect send by sending the following command:

```
REMOVE FX_SEND EFFECT <sampler-chan> <fx-send-id>
```

Where <sampler-chan> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command, <fx-send-id> reflects the numerical ID of the effect send entity as returned by the **"CREATE FX_SEND"** or **"LIST FX_SENDS"** command.

After the destination effect has been removed from the effect send, the audio signal of the effect send will be routed directly to the audio output device, according to the audio channel routing setting of the effect send.

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

```
C: "REMOVE FX_SEND EFFECT 0 0"

S: "OK"
```

6.4.37. Altering effect send's MIDI controller

TOC

The front-end can alter the MIDI controller of an effect send entity by sending the following command:

SET FX_SEND MIDI_CONTROLLER <sampler-chan> <fx-send-id> <midi-ctrl>

Where <sampler-chan> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command, <fx-send-id> reflects the numerical ID of the effect send entity as returned by the **"CREATE FX_SEND"** or **"LIST FX_SENDS"** command and <midi-ctrl> reflects the MIDI controller which shall be able to modify the effect send's send level.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if MIDI controller was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

C: "SET FX_SEND MIDI_CONTROLLER 0 0 91"

S: "OK"

6.4.38. Altering effect send's send level

TOC

The front-end can alter the current send level of an effect send entity by sending the following command:

SET FX_SEND LEVEL <sampler-chan> <fx-send-id> <volume>

Where <sampler-chan> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command, <fx-send-id> reflects the numerical ID of the effect send entity as returned by the **"CREATE FX_SEND"** or **"LIST FX_SENDS"** command and <volume> is an optionally dotted positive number (a value smaller than 1.0 means attenuation, whereas a value greater than 1.0 means amplification) reflecting the new send level.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if new send level was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

C: "SET FX_SEND LEVEL 0 0 0.15"

S: "OK"

TOC

6.4.39. Sending MIDI messages to sampler channel

The front-end can send MIDI events to a specific sampler channel by sending the following command:

SEND CHANNEL MIDI_DATA <midi-msg> <sampler-chan> <arg1> <arg2>

Where <sampler-chan> is the sampler channel number as returned by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command, <arg1> and <arg2> arguments depend on the <midi-msg> argument, which specifies the MIDI message type. Currently, the following MIDI messages are supported:

"NOTE_ON" -

For turning on MIDI notes, where <arg1> specifies the key number and <arg2> the velocity as described in the MIDI specification.

"NOTE_OFF" -

For turning a currently playing MIDI note off, where <arg1> specifies the key number and <arg2> the velocity as described in the MIDI specification.

"CC" -

For changing a MIDI controller, where <arg1> specifies the controller number and <arg2> the new value of the controller as described in the Control Change section of the MIDI specification.

CAUTION: This command is provided for implementations of virtual MIDI keyboards and no realtime guarantee whatsoever will be made!

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Example:

C: "SEND CHANNEL MIDI_DATA NOTE_ON 0 56 112"

S: "OK"

TOC

6.4.40. Resetting a sampler channel

The front-end 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:<warning-code>:<warning-message>" -

if channel was reset, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

6.5. Controlling connection

TOC

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

6.5.1. Register front-end for receiving event messages

TOC

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

SUBSCRIBE <event-id>

where <event-id> will be replaced by the respective event that client wants to subscribe to.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if registration succeeded, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

6.5.2. Unregister front-end for not receiving event messages

TOC

The front-end can unregister itself if it doesn't want to receive event messages anymore by sending the following command:

UNSUBSCRIBE <event-id>

Where <event-id> will be replaced by the respective event that client doesn't want to receive anymore.

Possible Answers:

"OK" -

on success

"WRN:<warning-code>:<warning-message>" -

if unregistration succeeded, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

6.5.3. 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" -

usually

"ERR:<error-code>:<error-message>" -

on syntax error, e.g. non boolean value

Examples:

6.5.4. 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 front-end implementation.

6.6. Global commands

The following commands have global impact on the sampler.

6.6.1. Current number of active voices

The front-end can ask for the current number of active voices on the sampler by sending the following command:

```
GET TOTAL_VOICE_COUNT
```

Possible Answers:

LinuxSampler will answer by returning the number of all active voices on the sampler.

6.6.2. Maximum amount of active voices

The front-end can ask for the maximum number of active voices by sending the following command:

```
GET TOTAL_VOICE_COUNT_MAX
```

Possible Answers:

LinuxSampler will answer by returning the maximum number of active voices.

6.6.3. Current number of active disk streams

The front-end can ask for the current number of active disk streams on the sampler by sending the following command:

```
GET TOTAL_STREAM_COUNT
```

Possible Answers:

LinuxSampler will answer by returning the number of all active disk streams on the sampler.

6.6.4. Reset sampler

The front-end can reset the whole sampler by sending the following command:

```
RESET
```

Possible Answers:

"OK" -

always

Examples:

6.6.5. General sampler informations

TOC

The client can ask for general informations about the LinuxSampler instance by sending the following command:

> GET SERVER INFO

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 information category. At the moment the following categories are defined:
>
> DESCRIPTION -
>
> > arbitrary textual description about the sampler (note that the character string may contain **escape sequences**)
>
> VERSION -
>
> > version of the sampler
>
> PROTOCOL_VERSION -
>
> > version of the LSCP specification the sampler complies with (see **Section 2** for details)
>
> INSTRUMENTS_DB_SUPPORT -
>
> > either yes or no, specifies whether the sampler is build with instruments database support.

The mentioned fields above don't have to be in particular order. Other fields might be added in future.

Example:

```
C: "GET SERVER INFO"

S: "DESCRIPTION: LinuxSampler - modular, streaming capable sampler"

   "VERSION: 1.0.0.svn23"

   "PROTOCOL_VERSION: 1.5"

   "INSTRUMENTS_DB_SUPPORT: no"

   "."
```

6.6.6. Getting global volume attenuation

TOC

The client can ask for the current global sampler-wide volume attenuation by sending the following command:

> GET VOLUME

Possible Answers:

> The sampler will always answer by returning the optional dotted floating point coefficient, reflecting the current global volume attenuation.

Note: it is up to the respective sampler engine whether to obey that global volume parameter or not, but in general all engines SHOULD use this parameter.

6.6.7. Setting global volume attenuation

The client can alter the current global sampler-wide volume attenuation by sending the following command:

```
SET VOLUME <volume>
```

Where <volume> should be replaced by the optional dotted floating point value, reflecting the new global volume parameter. This value might usually be in the range between 0.0 and 1.0, that is for attenuating the overall volume.

Possible Answers:

> "OK" -
>
> > on success
>
> "WRN:<warning-code>:<warning-message>" -
>
> > if the global volume was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message
>
> "ERR:<error-code>:<error-message>" -
>
> > in case it failed, providing an appropriate error code and error message

6.6.8. Getting global voice limit

The client can ask for the current global sampler-wide limit for maximum voices by sending the following command:

```
GET VOICES
```

Possible Answers:

> LinuxSampler will answer by returning the number for the current limit of maximum voices.

The voice limit setting defines how many voices should maximum be processed by the sampler at the same time. If the user triggers new notes which would exceed that voice limit, the sampler engine will react by stealing old voices for those newly triggered notes. Note that the amount of voices triggered by a new note can be larger than one and is dependent to the respective instrument and probably further criterias.

6.6.9. Setting global voice limit

The client can alter the current global sampler-wide limit for maximum voices by sending the following command:

> SET VOICES <max-voices>

Where <max-voices> should be replaced by the integer value, reflecting the new global amount limit of maximum voices. This value has to be larger than 0.

Possible Answers:

> "OK" -
>
> > on success
>
> "WRN:<warning-code>:<warning-message>" -
>
> > if the voice limit was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message
>
> "ERR:<error-code>:<error-message>" -
>
> > in case it failed, providing an appropriate error code and error message

Note: the given value will be passed to all sampler engine instances. The total amount of maximum voices on the running system might thus be as big as the given value multiplied by the current amount of engine instances.

Caution: when adjusting the voice limit, you SHOULD also adjust the disk stream limit respectively and vice versa.

TOC

6.6.10. Getting global disk stream limit

The client can ask for the current global sampler-wide limit for maximum disk streams by sending the following command:

> GET STREAMS

Possible Answers:

> LinuxSampler will answer by returning the number for the current limit of maximum disk streams.

The disk stream limit setting defines how many disk streams should maximum be processed by a sampler engine at the same time. The higher this value, the more memory (RAM) will be occupied, since every disk streams allocates a certain buffer size for being able to perform its streaming operations.

TOC

6.6.11. Setting global disk stream limit

The client can alter the current global sampler-wide limit for maximum disk streams by sending the following command:

> SET STREAMS <max-streams>

Where <max-streams> should be replaced by the integer value, reflecting the new global amount limit of maximum disk streams. This value has to be positive.

Possible Answers:

> "OK" -
>
> > on success
>
> "WRN:<warning-code>:<warning-message>" -
>
> > if the disk stream limit was set, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message
>
> "ERR:<error-code>:<error-message>" -
>
> > in case it failed, providing an appropriate error code and error message

Note: the given value will be passed to all sampler engine instances. The total amount of maximum disk streams on the running system might thus be as big as the given value multiplied by the current amount of engine instances.

Caution: when adjusting the disk stream limit, you SHOULD also adjust the voice limit respectively and vice versa.

6.7. MIDI Instrument Mapping

TOC

The MIDI protocol provides a way to switch between instruments by sending so called MIDI bank select and MIDI program change messages which are essentially just numbers. The following commands allow to actually map arbitrary MIDI bank select / program change numbers with real instruments.

The sampler allows to manage an arbitrary amount of MIDI instrument maps which define which instrument to load on which MIDI program change message.

By default, that is when the sampler is launched, there is no map, thus the sampler will simply ignore all program change messages. The front-end has to explicitly create at least one map, add entries to the map and tell the respective sampler channel(s) which MIDI instrument map to use, so the sampler knows how to react on a given program change message on the respective sampler channel, that is by switching to the respectively defined engine type and loading the respective instrument. See command **"SET CHANNEL MIDI_INSTRUMENT_MAP"** for how to assign a MIDI instrument map to a sampler channel.

Also note per MIDI specification a bank select message does not cause to switch to another instrument. Instead when receiving a bank select message the bank value will be stored and a subsequent program change message (which may occur at any time) will finally cause the sampler to switch to the respective instrument as reflected by the current MIDI instrument map.

6.7.1. Create a new MIDI instrument map

TOC

The front-end can add a new MIDI instrument map by sending the following command:

```
ADD MIDI_INSTRUMENT_MAP [<name>]
```

Where <name> is an optional argument allowing to assign a custom name to the new map. MIDI instrument Map names do not have to be unique, but MUST be encapsulated into apostrophes and support escape sequences as described in chapter "**Character Set and Escape Sequences**".

Possible Answers:

"OK[<map>]" -

in case a new MIDI instrument map could be added, where <map> reflects the unique ID of the newly created MIDI instrument map

"ERR:<error-code>:<error-message>" -

when a new map could not be created, which might never occur in practice

Examples:

C: "ADD MIDI_INSTRUMENT_MAP 'Standard Map'"

S: "OK[0]"

C: "ADD MIDI_INSTRUMENT_MAP 'Standard Drumkit'"

S: "OK[1]"

C: "ADD MIDI_INSTRUMENT_MAP"

S: "OK[5]"

TOC

6.7.2. Delete one particular or all MIDI instrument maps

The front-end can delete a particular MIDI instrument map by sending the following command:

REMOVE MIDI_INSTRUMENT_MAP <map>

Where <map> reflects the unique ID of the map to delete as returned by the **"LIST MIDI_INSTRUMENT_MAPS"** command.

The front-end can delete all MIDI instrument maps by sending the following command:

REMOVE MIDI_INSTRUMENT_MAP ALL

Possible Answers:

"OK" -

in case the map(s) could be deleted

"ERR:<error-code>:<error-message>" -

when the given map does not exist

Examples:

C: "REMOVE MIDI_INSTRUMENT_MAP 0"

S: "OK"

C: "REMOVE MIDI_INSTRUMENT_MAP ALL"

S: "OK"

6.7.3. Get amount of existing MIDI instrument maps

TOC

The front-end can retrieve the current amount of MIDI instrument maps by sending the following command:

```
GET MIDI_INSTRUMENT_MAPS
```

Possible Answers:

> The sampler will answer by returning the current number of MIDI instrument maps.

Example:

```
C: "GET MIDI_INSTRUMENT_MAPS"

S: "2"
```

6.7.4. Getting all created MIDI instrument maps

TOC

The number of MIDI instrument maps can change on runtime. To get the current list of MIDI instrument maps, the front-end can send the following command:

```
LIST MIDI_INSTRUMENT_MAPS
```

Possible Answers:

> The sampler will answer by returning a comma separated list with all MIDI instrument maps' numerical IDs.

Example:

```
C: "LIST MIDI_INSTRUMENT_MAPS"

S: "0,1,5,12"
```

6.7.5. Getting MIDI instrument map information

TOC

The front-end can ask for the current settings of a MIDI instrument map by sending the following command:

```
GET MIDI_INSTRUMENT_MAP INFO <map>
```

Where <map> is the numerical ID of the map the front-end is interested in as returned by the **"LIST MIDI_INSTRUMENT_MAPS"** command.

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:

NAME -

custom name of the given map, which does not have to be unique (note that this character string may contain **escape sequences**)

DEFAULT -

either true or false, defines whether this map is the default map

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

Example:

C: "GET MIDI_INSTRUMENT_MAP INFO 0"

S: "NAME: Standard Map"

"DEFAULT: true"

"."

6.7.6. Renaming a MIDI instrument map

TOC

The front-end can alter the custom name of a MIDI instrument map by sending the following command:

SET MIDI_INSTRUMENT_MAP NAME <map> <name>

Where <map> is the numerical ID of the map and <name> the new custom name of the map, which does not have to be unique (name MUST be encapsulated into apostrophes and supports escape sequences as described in chapter "**Character Set and Escape Sequences**").

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case the given map does not exist

Example:

C: "SET MIDI_INSTRUMENT_MAP NAME 0 'Foo instruments'"

S: "OK"

6.7.7. Create or replace a MIDI instrument map entry

TOC

The front-end can create a new or replace an existing entry in a sampler's MIDI instrument map by sending the following command:

MAP MIDI_INSTRUMENT [NON_MODAL] <map> <midi_bank> <midi_prog> <engine_name> <filename> <instrument_index> <volume_value> [<instr_load_mode>]

[<name>]

Where <map> is the numeric ID of the map to alter, <midi_bank> is an integer value between 0..16383 reflecting the MIDI bank select index, <midi_prog> an integer value between 0..127 reflecting the MIDI program change index, <engine_name> a sampler engine name as returned by the **"LIST AVAILABLE_ENGINES"** command (not encapsulated into apostrophes), <filename> the name of the instrument's file to be deployed (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**"), <instrument_index> the index (integer value) of the instrument within the given file, <volume_value> reflects the master volume of the instrument as optionally dotted number (where a value < 1.0 means attenuation and a value > 1.0 means amplification). This parameter easily allows to adjust the volume of all intruments within a custom instrument map without having to adjust their instrument files. The OPTIONAL <instr_load_mode> argument defines the life time of the instrument, that is when the instrument should be loaded, when freed and has exactly the following possibilities:

"ON_DEMAND" -

> The instrument will be loaded when needed, that is when demanded by at least one sampler channel. It will immediately be freed from memory when not needed by any sampler channel anymore.

"ON_DEMAND_HOLD" -

> The instrument will be loaded when needed, that is when demanded by at least one sampler channel. It will be kept in memory even when not needed by any sampler channel anymore. Instruments with this mode are only freed when the sampler is reset or all mapping entries with this mode (and respective instrument) are explicitly changed to "ON_DEMAND" and no sampler channel is using the instrument anymore.

"PERSISTENT" -

> The instrument will immediately be loaded into memory when this mapping command is sent and the instrument is kept all the time. Instruments with this mode are only freed when the sampler is reset or all mapping entries with this mode (and respective instrument) are explicitly changed to "ON_DEMAND" and no sampler channel is using the instrument anymore.

not supplied -

> In case there is no <instr_load_mode> argument given, it will be up to the InstrumentManager to decide which mode to use. Usually it will use "ON_DEMAND" if an entry for the given instrument does not exist in the InstrumentManager's list yet, otherwise if an entry already exists, it will simply stick with the mode currently reflected by the already existing entry, that is it will not change the mode.

The <instr_load_mode> argument thus allows to define an appropriate strategy (low memory consumption vs. fast instrument switching) for each instrument individually. Note, the following restrictions apply to this argument: "ON_DEMAND_HOLD" and "PERSISTENT" have to be supported by the respective sampler engine (which is technically the case when the engine provides an InstrumentManager for its format). If this is not the case the argument will automatically fall back to the default value "ON_DEMAND". Also the load mode of one instrument may automatically change the laod mode of other instrument(s), i.e. because the instruments are part of the same file and the engine does not allow a way to manage load modes for them individually. Due to this, in case the frontend shows the load modes of entries, the frontend should retrieve the actual mode by i.e. sending **"GET**

MIDI_INSTRUMENT INFO" command(s). Finally the OPTIONAL <name> argument allows to set a custom name (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**") for the mapping entry, useful for frontends for displaying an appropriate name for mapped instruments (using **"GET MIDI_INSTRUMENT INFO"**).

By default, "MAP MIDI_INSTRUMENT" commands block until the mapping is completely established in the sampler. The OPTIONAL "NON_MODAL" argument however causes the respective "MAP MIDI_INSTRUMENT" command to return immediately, that is to let the sampler establish the mapping in the background. So this argument might be especially useful for mappings with a "PERSISTENT" type, because these have to load the respective instruments immediately and might thus block for a very long time. It is recommended however to use the OPTIONAL "NON_MODAL" argument only if really necessary, because it has the following drawbacks: as "NON_MODAL" instructions return immediately, they may not necessarily return an error i.e. when the given instrument file turns out to be corrupt, beside that subsequent commands in a LSCP instruction sequence might fail, because mandatory mappings are not yet completed.

Possible Answers:

"OK" -

usually

"ERR:<error-code>:<error-message>" -

when the given map or engine does not exist or a value is out of range

Examples:

C: "MAP MIDI_INSTRUMENT 0 3 0 gig '/usr/share/Steinway D.gig' 0 0.8 PERSISTENT"

S: "OK"

C: "MAP MIDI_INSTRUMENT 0 4 50 gig '/home/john/foostrings.gig' 7 1.0"

S: "OK"

C: "MAP MIDI_INSTRUMENT 0 0 0 gig '/usr/share/piano.gig' 0 1.0 'Normal Piano'"

S: "OK"

C: "MAP MIDI_INSTRUMENT 0 1 0 gig '/usr/share/piano.gig' 0 0.25 'Silent Piano'"

S: "OK"

C: "MAP MIDI_INSTRUMENT NON_MODAL 1 8 120 gig '/home/joe/foodrums.gig' 0 1.0 PERSISTENT 'Foo Drumkit'"

S: "OK"

TOC

6.7.8. Getting amount of MIDI instrument map entries

The front-end can query the amount of currently existing entries in a MIDI instrument map by sending the following command:

GET MIDI_INSTRUMENTS <map>

The front-end can query the amount of currently existing entries in all MIDI instrument maps by sending the following command:

```
GET MIDI_INSTRUMENTS ALL
```

Possible Answers:

> The sampler will answer by sending the current number of entries in the MIDI instrument map(s).

Example:

```
C: "GET MIDI_INSTRUMENTS 0"

S: "234"

C: "GET MIDI_INSTRUMENTS ALL"

S: "954"
```

6.7.9. Getting indeces of all entries of a MIDI instrument map

TOC

The front-end can query a list of all currently existing entries in a certain MIDI instrument map by sending the following command:

```
LIST MIDI_INSTRUMENTS <map>
```

Where <map> is the numeric ID of the MIDI instrument map.

The front-end can query a list of all currently existing entries of all MIDI instrument maps by sending the following command:

```
LIST MIDI_INSTRUMENTS ALL
```

Possible Answers:

> The sampler will answer by sending a comma separated list of map ID - MIDI bank - MIDI program triples, where each triple is encapsulated into curly braces. The list is returned in one single line. Each triple just reflects the key of the respective map entry, thus subsequent **"GET MIDI_INSTRUMENT INFO"** command(s) are necessary to retrieve detailed informations about each entry.

Example:

```
C: "LIST MIDI_INSTRUMENTS 0"

S: "{0,0,0},{0,0,1},{0,0,3},{0,1,4},{1,127,127}"
```

6.7.10. Remove an entry from the MIDI instrument map

TOC

The front-end can delete an entry from a MIDI instrument map by sending the following command:

```
UNMAP MIDI_INSTRUMENT <map> <midi_bank> <midi_prog>
```

Where <map> is the numeric ID of the MIDI instrument map, <midi_bank> is an integer value between 0..16383 reflecting the MIDI bank value and <midi_prog> an integer value between 0..127 reflecting the MIDI program value of the map's entrie's key index triple.

Possible Answers:

"OK" -

usually

"ERR:<error-code>:<error-message>" -

when index out of bounds

Example:

C: "UNMAP MIDI_INSTRUMENT 0 2 127"

S: "OK"

TOC

6.7.11. Get current settings of MIDI instrument map entry

The front-end can retrieve the current settings of a certain instrument map entry by sending the following command:

GET MIDI_INSTRUMENT INFO <map> <midi_bank> <midi_prog>

Where <map> is the numeric ID of the MIDI instrument map, <midi_bank> is an integer value between 0..16383 reflecting the MIDI bank value, <midi_bank> and <midi_prog> an integer value between 0..127 reflecting the MIDI program value of the map's entrie's key index triple.

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:

"NAME" -

Name for this MIDI instrument map entry (if defined). This name shall be used by frontends for displaying a name for this mapped instrument. It can be set and changed with the **"MAP MIDI_INSTRUMENT"** command and does not have to be unique. (note that this character string may contain **escape sequences**)

"ENGINE_NAME" -

Name of the engine to be deployed for this instrument.

"INSTRUMENT_FILE" -

File name of the instrument (note that this path may contain **escape sequences**).

"INSTRUMENT_NR" -

Index of the instrument within the file.

"INSTRUMENT_NAME" -

Name of the loaded instrument as reflected by its file. In contrast to the

"NAME" field, the "INSTRUMENT_NAME" field cannot be changed (note that this character string may contain **escape sequences**).

"LOAD_MODE" -

Life time of instrument (see **"MAP MIDI_INSTRUMENT"** for details about this setting).

"VOLUME" -

master volume of the instrument as optionally dotted number (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 MIDI_INSTRUMENT INFO 1 45 120"

S: "NAME: Drums for Foo Song"

"ENGINE_NAME: GigEngine"

"INSTRUMENT_FILE: /usr/share/joesdrumkit.gig"

"INSTRUMENT_NR: 0"

"INSTRUMENT_NAME: Joe's Drumkit"

"LOAD_MODE: PERSISTENT"

"VOLUME: 1.0"

"."

6.7.12. Clear MIDI instrument map

TOC

The front-end can clear a whole MIDI instrument map, that is delete all its entries by sending the following command:

CLEAR MIDI_INSTRUMENTS <map>

Where <map> is the numeric ID of the map to clear.

The front-end can clear all MIDI instrument maps, that is delete all entries of all maps by sending the following command:

CLEAR MIDI_INSTRUMENTS ALL

The command "CLEAR MIDI_INSTRUMENTS ALL" does not delete the maps, only their entries, thus the map's settings like custom name will be preservevd.

Possible Answers:

"OK" -

always

Examples:

C: "CLEAR MIDI_INSTRUMENTS 0"

S: "OK"

C: "CLEAR MIDI_INSTRUMENTS ALL"

S: "OK"

6.8. Managing Instruments Database

TOC

The following commands describe how to use and manage the instruments database.

Notice:

> All command arguments representing a path or instrument/directory name support escape sequences as described in chapter "**Character Set and Escape Sequences**".
>
> All occurrences of a forward slash in instrument and directory names are escaped with its hex (\x2f) or octal (\057) escape sequence.

6.8.1. Creating a new instrument directory

TOC

The front-end can add a new instrument directory to the instruments database by sending the following command:

```
ADD DB_INSTRUMENT_DIRECTORY <dir>
```

Where <dir> is the absolute path name of the directory to be created (encapsulated into apostrophes).

Possible Answers:

"OK" -

> on success

"ERR:<error-code>:<error-message>" -

> when the directory could not be created, which can happen if the directory already exists or the name contains not allowed symbols

Examples:

```
C: "ADD DB_INSTRUMENT_DIRECTORY '/Piano Collection'"

S: "OK"
```

6.8.2. Deleting an instrument directory

TOC

The front-end can delete a particular instrument directory from the instruments database by sending the following command:

```
REMOVE DB_INSTRUMENT_DIRECTORY [FORCE] <dir>
```

Where <dir> is the absolute path name of the directory to delete. The optional FORCE argument can be used to force the deletion of a non-empty directory and all its content.

Possible Answers:

"OK" -

if the directory is deleted successfully

"ERR:<error-code>:<error-message>" -

if the given directory does not exist, or if trying to delete a non-empty directory, without using the FORCE argument.

Examples:

C: "REMOVE DB_INSTRUMENT_DIRECTORY FORCE '/Piano Collection'"

S: "OK"

6.8.3. Getting amount of instrument directories

TOC

The front-end can retrieve the current amount of directories in a specific directory by sending the following command:

GET DB_INSTRUMENT_DIRECTORIES [RECURSIVE] <dir>

Where <dir> should be replaced by the absolute path name of the directory. If RECURSIVE is specified, the number of all directories, including those located in subdirectories of the specified directory, will be returned.

Possible Answers:

The current number of instrument directories in the specified directory.

"ERR:<error-code>:<error-message>" -

if the given directory does not exist.

Example:

C: "GET DB_INSTRUMENT_DIRECTORIES '/'"

S: "2"

6.8.4. Listing all directories in specific directory

TOC

The front-end can retrieve the current list of directories in specific directory by sending the following command:

LIST DB_INSTRUMENT_DIRECTORIES [RECURSIVE] <dir>

Where <dir> should be replaced by the absolute path name of the directory. If RECURSIVE is specified, the absolute path names of all directories, including those located in subdirectories of the specified directory, will be returned.

Possible Answers:

A comma separated list of all instrument directories (encapsulated into apostrophes) in the specified directory.

"ERR:<error-code>:<error-message>" -

if the given directory does not exist.

Example:

C: "LIST DB_INSTRUMENT_DIRECTORIES '/'"

S: "'Piano Collection','Percussion Collection'"

C: "LIST DB_INSTRUMENT_DIRECTORIES RECURSIVE '/'"

S: "'/Piano Collection','/Piano Collection/Acoustic','/Piano Collection/Acoustic/New','/Percussion Collection'"

TOC

6.8.5. Getting instrument directory information

The front-end can ask for the current settings of an instrument directory by sending the following command:

GET DB_INSTRUMENT_DIRECTORY INFO <dir>

Where <dir> should be replaced by the absolute path name of the directory the front-end 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:

DESCRIPTION -

A brief description of the directory content. Note that the character string may contain **escape sequences**.

CREATED -

The creation date and time of the directory, represented in "YYYY-MM-DD HH:MM:SS" format

MODIFIED -

The date and time of the last modification of the directory, represented in "YYYY-MM-DD HH:MM:SS" format

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

Example:

C: "GET DB_INSTRUMENT_DIRECTORY INFO '/Piano Collection'"

S: "DESCRIPTION: Piano collection of instruments in GigaSampler format."

"CREATED: 2007-02-05 10:23:12"

"MODIFIED: 2007-04-07 12:50:21"

"."

TOC

6.8.6. Renaming an instrument directory

The front-end can alter the name of a specific instrument directory by sending the following command:

SET DB_INSTRUMENT_DIRECTORY NAME <dir> <name>

Where <dir> is the absolute path name of the directory and <name> is the new name for that directory.

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case the given directory does not exists, or if a directory with name equal to the new name already exists.

Example:

C: "SET DB_INSTRUMENT_DIRECTORY NAME '/Piano Collection/Acustic' 'Acoustic'"

S: "OK"

TOC

6.8.7. Moving an instrument directory

The front-end can move a specific instrument directory by sending the following command:

MOVE DB_INSTRUMENT_DIRECTORY <dir> <dst>

Where <dir> is the absolute path name of the directory to move and <dst> is the location where the directory will be moved to.

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case a given directory does not exists, or if a directory with name equal to the name of the specified directory already exists in the destination directory. Error is also thrown when trying to move a directory to a subdirectory of itself.

Example:

C: "MOVE DB_INSTRUMENT_DIRECTORY '/Acoustic' '/Piano Collection/Acoustic'"

S: "OK"

6.8.8. Copying instrument directories

The front-end can copy a specific instrument directory by sending the following command:

COPY DB_INSTRUMENT_DIRECTORY <dir> <dst>

Where <dir> is the absolute path name of the directory to copy and <dst> is the location where the directory will be copied to.

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case a given directory does not exists, or if a directory with name equal to the name of the specified directory already exists in the destination directory. Error is also thrown when trying to copy a directory to a subdirectory of itself.

Example:

C: "COPY DB_INSTRUMENT_DIRECTORY '/Piano Collection/Acoustic' '/Acoustic/Pianos'"

S: "OK"

6.8.9. Changing the description of directory

The front-end can alter the description of a specific instrument directory by sending the following command:

SET DB_INSTRUMENT_DIRECTORY DESCRIPTION <dir> <desc>

Where <dir> is the absolute path name of the directory and <desc> is the new description for the directory (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**").

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case the given directory does not exists.

Example:

C: "SET DB_INSTRUMENT_DIRECTORY DESCRIPTION '/Piano Collection' 'A collection of piano instruments in various format.'"

S: "OK"

6.8.10. Finding directories

TOC

The front-end can search for directories in specific directory by sending the following command:

FIND DB_INSTRUMENT_DIRECTORIES [NON_RECURSIVE] <dir> <criteria-list>

Where <dir> should be replaced by the absolute path name of the directory to search in. If NON_RECURSIVE is specified, the directories located in subdirectories of the specified directory will not be searched. <criteria-list> is a list of search criterias in form of "key1=val1 key2=val2 ...". The following criterias are allowed:

NAME='<search-string>'

Restricts the search to directories, which names satisfy the supplied search string (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**").

CREATED='[<date-after>]..[<date-before>]'

Restricts the search to directories, which creation date satisfies the specified period, where <date-after> and <date-before> are in "YYYY-MM-DD HH:MM:SS" format. If <date-after> is omitted the search is restricted to directories created before <date-before>. If <date-before> is omitted, the search is restricted to directories created after <date-after>.

MODIFIED='[<date-after>]..[<date-before>]'

Restricts the search to directories, which date of last modification satisfies the specified period, where <date-after> and <date-before> are in "YYYY-MM-DD HH:MM:SS" format. If <date-after> is omitted the search is restricted to directories, which are last modified before <date-before>. If <date-before> is omitted, the search is restricted to directories, which are last modified after <date-after>.

DESCRIPTION='<search-string>'

Restricts the search to directories with description that satisfies the supplied search string (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**").

Where <search-string> is either a regular expression, or a word list separated with spaces for OR search and with '+' for AND search.

Possible Answers:

A comma separated list with the absolute path names (encapsulated into apostrophes) of all directories in the specified directory that satisfy the supplied search criterias.

"ERR:<error-code>:<error-message>" -

if the given directory does not exist.

Example:

C: "FIND DB_INSTRUMENT_DIRECTORIES '/' NAME='Piano'"

S: "'/Piano Collection'"

C: "FIND DB_INSTRUMENT_DIRECTORIES '/' CREATED='..2007-04-01 09:30:13'"

S: "'/Piano Collection','/Percussions'"

TOC

6.8.11. Adding instruments to the instruments database

The front-end can add one or more instruments to the instruments database by sending the following command:

ADD DB_INSTRUMENTS [NON_MODAL] [<mode>[FILE_AS_DIR]] <db_dir> <file_path> [<instr_index>]

Where <db_dir> is the absolute path name of a directory (encapsulated into apostrophes) in the instruments database in which only the new instruments (that are not already in the database) will be added, <file_path> is the absolute path name of a file or directory in the file system (encapsulated into apostrophes). In case an instrument file is supplied, only the instruments in the specified file will be added to the instruments database. If the optional <instr_index> (the index of the instrument within the given file) is supplied too, then only the specified instrument will be added. In case a directory is supplied, the instruments in that directory will be added. The OPTIONAL <mode> argument is only applied when a directory is provided as <file_path> and specifies how the scanning will be done and has exactly the following possibilities:

"RECURSIVE" -

All instruments will be processed, including those in the subdirectories, and the respective subdirectory tree structure will be recreated in the instruments database

"NON_RECURSIVE" -

Only the instruments in the specified directory will be added, the instruments in the subdirectories will not be processed.

"FLAT" -

All instruments will be processed, including those in the subdirectories, but the respective subdirectory structure will not be recreated in the instruments database. All instruments will be added directly in the specified database directory.

If FILE_AS_DIR argument is supplied, all instruments in an instrument file will be added to a separate directory in the instruments database, which name will be the name of the instrument file with the file extension stripped off.

The difference between regular and NON_MODAL versions of the command is that the regular command returns when the scanning is finished while NON_MODAL version returns immediately and a background process is launched. The **GET DB_INSTRUMENTS_JOB INFO** command can be used to monitor the scanning progress.

Possible Answers:

"OK" -

on success when NON_MODAL is not supplied

"OK[<job-id>]" -

on success when NON_MODAL is supplied, where <job-id> is a numerical ID used to obtain status information about the job progress. See **GET DB_INSTRUMENTS_JOB INFO**

"ERR:<error-code>:<error-message>" -

if an invalid path is specified.

Examples:

C: "ADD DB_INSTRUMENTS '/Piano Collection' '/home/me/gigs/PMI Bosendorfer 290.gig' 0"

S: "OK"

TOC

6.8.12. Removing an instrument

The front-end can remove a particular instrument from the instruments database by sending the following command:

REMOVE DB_INSTRUMENT <instr_path>

Where <instr_path> is the absolute path name (in the instruments database) of the instrument to remove.

Possible Answers:

"OK" -

if the instrument is removed successfully

"ERR:<error-code>:<error-message>" -

if the given path does not exist or is a directory.

Examples:

C: "REMOVE DB_INSTRUMENT '/Piano Collection/Bosendorfer 290'"

S: "OK"

TOC

6.8.13. Getting amount of instruments

The front-end can retrieve the current amount of instruments in a specific directory by sending the following command:

GET DB_INSTRUMENTS [RECURSIVE] <dir>

Where <dir> should be replaced by the absolute path name of the directory. If RECURSIVE is specified, the number of all instruments, including those located in subdirectories of the specified directory, will be returned.

Possible Answers:

The current number of instruments in the specified directory.

"ERR:<error-code>:<error-message>" -

if the given directory does not exist.

Example:

```
C: "GET DB_INSTRUMENTS '/Piano Collection'"
```

```
S: "2"
```

6.8.14. Listing all instruments in specific directory

TOC

The front-end can retrieve the current list of instruments in specific directory by sending the following command:

```
LIST DB_INSTRUMENTS [RECURSIVE] <dir>
```

Where <dir> should be replaced by the absolute path name of the directory. If RECURSIVE is specified, the absolute path names of all instruments, including those located in subdirectories of the specified directory, will be returned.

Possible Answers:

A comma separated list of all instruments (encapsulated into apostrophes) in the specified directory.

"ERR:<error-code>:<error-message>" -

if the given directory does not exist.

Example:

```
C: "LIST DB_INSTRUMENTS '/Piano Collection'"
```

```
S: "'Bosendorfer 290','Steinway D'"
```

```
C: "LIST DB_INSTRUMENTS RECURSIVE '/Piano Collection'"
```

```
S: "'/Piano Collection/Bosendorfer 290','/Piano Collection/Steinway D','/Piano
Collection/Lite/Free Piano'"
```

6.8.15. Getting instrument information

TOC

The front-end can ask for the current settings of an instrument by sending the following command:

```
GET DB_INSTRUMENT INFO <instr_path>
```

Where <instr_path> should be replaced by the absolute path name of the instrument the front-end 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:

INSTRUMENT_FILE -

> File name of the instrument. Note that the character string may contain **escape sequences**.

INSTRUMENT_NR -

> Index of the instrument within the file.

FORMAT_FAMILY -

> The format family of the instrument.

FORMAT_VERSION -

> The format version of the instrument.

SIZE -

> The size of the instrument in bytes.

CREATED -

> The date and time when the instrument is added in the instruments database, represented in "YYYY-MM-DD HH:MM:SS" format

MODIFIED -

> The date and time of the last modification of the instrument's database settings, represented in "YYYY-MM-DD HH:MM:SS" format

DESCRIPTION -

> A brief description of the instrument. Note that the character string may contain **escape sequences**.

IS_DRUM -

> either true or false, determines whether the instrument is a drumkit or a chromatic instrument

PRODUCT -

> The product title of the instrument. Note that the character string may contain **escape sequences**.

ARTISTS -

> Lists the artist names. Note that the character string may contain **escape sequences**.

KEYWORDS -

> Provides a list of keywords that refer to the instrument.

Keywords are separated with semicolon and blank. Note that the character string may contain **escape sequences**.

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

Example:

```
C: "GET DB_INSTRUMENT INFO '/Piano Collection/Bosendorfer 290'"

S: "INSTRUMENT_FILE: /home/me/gigs/Bosendorfer 290.gig"

   "INSTRUMENT_NR: 0"

   "FORMAT_FAMILY: GIG"

   "FORMAT_VERSION: 2"

   "SIZE: 2050871870"

   "CREATED: 2007-02-05 10:23:12"

   "MODIFIED: 2007-04-07 12:50:21"

   "DESCRIPTION: "

   "IS_DRUM: false"

   "PRODUCT: GRANDIOSO Bosendorfer 290"

   "ARTISTS: Post Musical Instruments"

   "KEYWORDS: Bosendorfer"

   "."
```

6.8.16. Renaming an instrument

TOC

The front-end can alter the name of a specific instrument by sending the following command:

```
SET DB_INSTRUMENT NAME <instr> <name>
```

Where <instr> is the absolute path name of the instrument and <name> is the new name for that instrument.

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case the given instrument does not exists, or if an instrument with name equal to the new name already exists.

Example:

```
C: "SET DB_INSTRUMENT NAME '/Piano Collection/Bosendorfer' 'Bosendorfer 290'"

S: "OK"
```

TOC

6.8.17. Moving an instrument

The front-end can move a specific instrument to another directory by sending the following command:

```
MOVE DB_INSTRUMENT <instr> <dst>
```

Where <instr> is the absolute path name of the instrument to move and <dst> is the directory where the instrument will be moved to.

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case the given instrument does not exists, or if an instrument with name equal to the name of the specified instrument already exists in the destination directory.

Example:

```
C: "MOVE DB_INSTRUMENT '/Piano Collection/Bosendorfer 290' '/Piano
Collection/Acoustic'"

S: "OK"
```

TOC

6.8.18. Copying instruments

The front-end can copy a specific instrument to another directory by sending the following command:

```
COPY DB_INSTRUMENT <instr> <dst>
```

Where <instr> is the absolute path name of the instrument to copy and <dst> is the directory where the instrument will be copied to.

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case the given instrument does not exists, or if an instrument with name equal to the name of the specified instrument already exists in the destination directory.

Example:

```
C: "COPY DB_INSTRUMENT '/Piano Collection/Bosendorfer 290' '/Acoustic/Pianos/'"

S: "OK"
```

6.8.19. Changing the description of instrument

TOC

The front-end can alter the description of a specific instrument by sending the following command:

> SET DB_INSTRUMENT DESCRIPTION <instr> <desc>

Where <instr> is the absolute path name of the instrument and <desc> is the new description for the instrument (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**").

Possible Answers:

> "OK" -
>
> > on success
>
> "ERR:<error-code>:<error-message>" -
>
> > in case the given instrument does not exists.

Example:

> C: "SET DB_INSTRUMENT DESCRIPTION '/Piano Collection/Acoustic/Bosendorfer 290' 'No comment :)'"
>
> S: "OK"

6.8.20. Finding instruments

TOC

The front-end can search for instruments in specific directory by sending the following command:

> FIND DB_INSTRUMENTS [NON_RECURSIVE] <dir> <criteria-list>

Where <dir> should be replaced by the absolute path name of the directory to search in. If NON_RECURSIVE is specified, the directories located in subdirectories of the specified directory will not be searched. <criteria-list> is a list of search criterias in form of "key1=val1 key2=val2 ...". The following criterias are allowed:

NAME='<search-string>'

> Restricts the search to instruments, which names satisfy the supplied search string (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**").

SIZE=[<min>]..[<max>]

> Restricts the search to instruments, which size is in the specified range. If <min> is omitted, the search results are restricted to instruments with size less then or equal to <max>. If <max> is omitted, the search is restricted to instruments with size greater then or equal to <min>.

CREATED='[<date-after>]..[<date-before>]'

> Restricts the search to instruments, which creation date satisfies the specified period, where <date-after> and <date-before> are in "YYYY-MM-DD HH:MM:SS" format. If <date-after> is omitted the search is restricted to instruments created before <date-

before>. If <date-before> is omitted, the search is restricted to instruments created after <date-after>.

MODIFIED='[<date-after>]..[<date-before>]'

Restricts the search to instruments, which date of last modification satisfies the specified period, where <date-after> and <date-before> are in "YYYY-MM-DD HH:MM:SS" format. If <date-after> is omitted the search is restricted to instruments, which are last modified before <date-before>. If <date-before> is omitted, the search is restricted to instruments, which are last modified after <date-after>.

DESCRIPTION='<search-string>'

Restricts the search to instruments with description that satisfies the supplied search string (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**").

PRODUCT='<search-string>'

Restricts the search to instruments with product info that satisfies the supplied search string (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**").

ARTISTS='<search-string>'

Restricts the search to instruments with artists info that satisfies the supplied search string (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**").

KEYWORDS='<search-string>'

Restricts the search to instruments with keyword list that satisfies the supplied search string (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**").

IS_DRUM=true | false

Either true or false. Restricts the search to drum kits or chromatic instruments.

FORMAT_FAMILIES='<format-list>'

Restricts the search to instruments of the supplied format families, where <format-list> is a comma separated list of format families.

Where <search-string> is either a regular expression, or a word list separated with spaces for OR search and with '+' for AND search.

Possible Answers:

A comma separated list with the absolute path names (encapsulated into apostrophes) of all instruments in the specified directory that satisfy the supplied search criterias.

"ERR:<error-code>:<error-message>" -

if the given directory does not exist.

Example:

C: "FIND DB_INSTRUMENTS '/Piano Collection' NAME='bosendorfer+290'"

S: "'/Piano Collection/Bosendorfer 290'"

C: "FIND DB_INSTRUMENTS '/Piano Collection' CREATED='2007-04-01 09:30:13..'"

S: "'/Piano Collection/Bosendorfer 290','/Piano Collection/Steinway D'"

6.8.21. Getting job status information

TOC

The front-end can ask for the current status of a particular database instruments job by sending the following command:

GET DB_INSTRUMENTS_JOB INFO <job-id>

Where <job-id> should be replaced by the numerical ID of the job the front-end 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:

FILES_TOTAL -

The total number of files scheduled for scanning

FILES_SCANNED -

The current number of scanned files

SCANNING -

The absolute path name of the file which is currently being scanned

STATUS -

An integer value between 0 and 100 indicating the scanning progress percentage of the file which is currently being scanned

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

Example:

C: "GET DB_INSTRUMENTS_JOB INFO 2"

S: "FILES_TOTAL: 12"

"FILES_SCANNED: 7"

"SCANNING: /home/me/gigs/Bosendorfer 290.gig"

"STATUS: 42"

"."

6.8.22. Formatting the instruments database

The front-end can remove all instruments and directories and re-create the instruments database structure (e.g., in case of a database corruption) by sending the following command:

FORMAT INSTRUMENTS_DB

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

If the formatting of the instruments database failed.

6.8.23. Checking for lost instrument files

The front-end can retrieve the list of all instrument files in the instruments database that don't exist in the filesystem by sending the following command:

FIND LOST DB_INSTRUMENT_FILES

Possible Answers:

A comma separated list with the absolute path names (encapsulated into apostrophes) of all lost instrument files.

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message.

Example:

C: "FIND LOST DB_INSTRUMENT_FILES"

S: "'/gigs/Bosendorfer 290.gig','/gigs/Steinway D.gig','/gigs/Free Piano.gig'"

6.8.24. Replacing an instrument file

The front-end can substitute all occurrences of an instrument file in the instruments database with a new one by sending the following command:

SET DB_INSTRUMENT FILE_PATH <old_path> <new_path>

Where <old_path> is the absolute path name of the instrument file to substitute with <new_path>.

Possible Answers:

"OK" -

on success

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message.

Example:

C: "SET DB_INSTRUMENT FILE_PATH '/gigs/Bosendorfer 290.gig' '/gigs/pianos/Bosendorfer 290.gig'"

S: "OK"

6.9. Editing Instruments

TOC

The sampler allows to edit instruments while playing with the sampler by spawning an external (3rd party) instrument editor application for a given instrument. The 3rd party instrument editor applications have to place a respective plugin DLL file into the sampler's plugins directory. The sampler will automatically try to load all plugin DLLs in that directory on startup and only on startup!

At the moment there is only one command for this feature set, but this will most probably change in future.

6.9.1. Opening an appropriate instrument editor application

TOC

The front-end can request to open an appropriate instrument editor application by sending the following command:

EDIT CHANNEL INSTRUMENT <sampler-channel>

Where <sampler-channel> should be replaced by the number of the sampler channel as given by the **"ADD CHANNEL"** or **"LIST CHANNELS"** command.

The sampler will try to ask all registered instrument editors (or to be more specific: their sampler plugins) whether they are capable to handle the instrument on the given sampler channel. The sampler will simply use the first instrument editor application which replied with a positive answer and spawn that instrument editor application within the sampler's process and provide that application access to the instrument's data structures, so both applications can share and access the same instruments data at the same time, thus allowing to immediately hear changes with the sampler made by the instrument editor.

Note: consequently instrument editors are always spawned locally on the same machine where the sampler is running on!

Possible Answers:

"OK" -

when an appropriate instrument editor was launched

"WRN:<warning-code>:<warning-message>" -

when an appropriate instrument editor was launched, but there are noteworthy issues

"ERR:<error-code>:<error-message>" -

when an appropriate instrument editor could not be launched

Examples:

C: "EDIT CHANNEL INSTRUMENT 0"

S: "OK"

TOC

6.10. Managing Files

You can query detailed informations about files located at the same system where the sampler instance is running on. Using this command set allows to retrieve file informations even remotely from another machine.

TOC

6.10.1. Retrieving amount of instruments of a file

The front-end can retrieve the amount of instruments within a given instrument file by sending the following command:

GET FILE INSTRUMENTS <filename>

Where <filename> is the name of the instrument file (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**").

The sampler will try to ask all sampler engines, whether they support the given file and ask the first engine with a positive answer for the amount of instruments.

Possible Answers:

On success, the sampler will answer by returning the amount of instruments.

"ERR:<error-code>:<error-message>" -

if the file could not be handled

Examples:

C: "GET FILE INSTRUMENTS 'D:/Sounds/Foo.gig'"

S: "10"

TOC

6.10.2. Retrieving all instruments of a file

The front-end can retrieve a list of all instruments within a given instrument file by sending the following command:

LIST FILE INSTRUMENTS <filename>

Where <filename> is the name of the instrument file (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**").

The sampler will try to ask all sampler engines, whether they support the given file and ask the first engine with a positive answer for a list of IDs for the instruments in the given file.

Possible Answers:

On success, the sampler will answer by returning a comma separated list of instrument IDs.

"ERR:<error-code>:<error-message>" -

if the file could not be handled

Examples:

C: "LIST FILE INSTRUMENTS 'D:/Sounds/Foo.gig'"

S: "0,1,2,3,4,5,6,7,8,9"

6.10.3. Retrieving informations about one instrument in a file

TOC

The front-end can retrieve detailed informations about a specific instrument within a given instrument file by sending the following command:

GET FILE INSTRUMENT INFO <filename> <instr-id>

Where <filename> is the name of the instrument file (encapsulated into apostrophes, supporting escape sequences as described in chapter "**Character Set and Escape Sequences**") and <instr-id> is the numeric instrument ID as returned by the **"LIST FILE INSTRUMENTS"** command.

The sampler will try to ask all sampler engines, whether they support the given file and ask the first engine with a positive answer for informations about the specific instrument in the given file.

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:

NAME -

name of the instrument as stored in the instrument file

FORMAT_FAMILY -

name of the sampler format of the given instrument

FORMAT_VERSION -

version of the sampler format the instrumen is stored as

PRODUCT -

official product name of the instrument as stored in the file

ARTISTS -

artists / sample library vendor of the instrument

KEY_BINDINGS -

comma separated list of integer values representing the instrument's key mapping in the range between 0 .. 127, reflecting the analog meaning of the MIDI specification.

KEYSWITCH_BINDINGS -

comma separated list of integer values representing the instrument's keyswitch mapping in the range between 0 .. 127, reflecting the analog meaning of the MIDI specification.

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

Example:

```
C: "GET FILE INSTRUMENT INFO 'D:/Sounds/Foo.gig' 0"

S: "NAME: Lunatic Loops"

   "FORMAT_FAMILY: GIG"

   "FORMAT_VERSION: 3"

   "PRODUCT: The Backbone Bongo Beats"

   "ARTISTS: Jimmy the Fish"

   "."
```

6.11. Managing Effects

TOC

Audio effects (e.g. reverb, delay, compression) can be applied to the audio signals generated by the sampler. The sampler usually provides a set of internal audio effects for this task. The exact set of effects depends on the availability of third party effect plugins installed on the system where the sampler runs on.

At the moment only "send effects" are supported. Support for "insert effects" and "master effects" is planned to be added at a later point.

The following commands allow to retrieve the set of internal effects available to the sampler, detailed informations about those effects and to create and destroy instances of such effects. After an instance of an effect is created, the effect instance can be inserted into the audio signal path of the sampler, e.g. as send effect.

The sampler allows to create an arbitrary amount of so called send effect chains. Each effect chain can host an arbitrary amount of effect instances. The output of the first effect instance in an effect chain is fed to the input of the second effect instance of the chain and so on. So effects in one chain are processed sequentially. Send effect chains however are processed in parallel to other send effect chains. Audio signals of sampler channels are fed to send effects by creating FX sends to the respective sampler channel and assigning a destination send effect to that FX by using the **"SET FX_SEND EFFECT"** command. The latter allows to route the FX send to the beginning of a send effect chain, as well as directly to any other position of the send effect chain.

6.11.1. Retrieve amount of available effects

TOC

The front-end can retrieve the amount of internal effects, available to the sampler by sending the following command:

GET AVAILABLE_EFFECTS

Possible Answers:

The sampler will answer by returning the current number of effects available to the sampler.

Examples:

C: "GET AVAILABLE_EFFECTS"

S: "129"

6.11.2. Get list of available effects

TOC

The set of available internal effects can change at runtime. The front-end can retrieve the list of internal effects, available to the sampler by sending the following command:

LIST AVAILABLE_EFFECTS

Possible Answers:

The sampler will answer by returning a comma separated list with numerical IDs of effects. Note: the numercial ID of an effect is generated by the sampler for the current moment. The numerical ID of the same effect can change at runtime, e.g. when the user requests a rescan of available effect plugins.

Example:

C: "LIST AVAILABLE_EFFECTS"

S: "5,6,7,120,121,122,123,124"

6.11.3. Retrieving general information about an effect

TOC

The front-end can ask for general informations about an effect by sending the following command:

GET EFFECT INFO <effect-index>

Where <effect-index> is the numerical ID of an effect as returned by the **"LIST AVAILABLE_EFFECTS"** command.

Possible Answers:

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

SYSTEM -

name of the effect plugin system the effect is based on (e.g. "LADSPA")

MODULE -

module of the effect plugin system that contains this effect, the module is usually the dynamic-linked library (DLL) filename of the effect plugin, including full path (note that this filename may contain **escape sequences**)

NAME -

character string defining the unique name of the effect within its module (note that the character string may contain **escape sequences**)

DESCRIPTION -

human readable name of the effect, intended to be displayed in user interfaces (note that the character string may contain **escape sequences**)

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

Example:

C: "GET EFFECT INFO 121"

S: "SYSTEM: LADSPA"

"MODULE: /usr/lib/ladspa/lowpass_iir_1891.so"

"NAME: lowpass_iir"

"DESCRIPTION: Glame Lowpass Filter"

"."

6.11.4. Creating an instance of an effect by its portable ID

TOC

The front-end can spawn an instance of the desired effect by sending the following command:

CREATE EFFECT_INSTANCE <effect-system> <module> <effect-name>

Where <effect-system> is the "SYSTEM" field, <module> the "MODULE" field and <effect-name> the "NAME" field as returned by the **"GET EFFECT INFO"** command. The filename of argument <module> and the character string of argument <effect-name> may contain **escape sequences**.

The sampler will try to load the requested effect and to create an instance of it. To allow loading the same effect on a different machine, probably even running a completely different operating system (e.g. Linux vs. Windows), the sampler tries to match <module> "softly". That means it first tries to find an effect that exactly matches the given <module> argument. If there is no exact match, the sampler will try to lower the restrictions on matching the <module> argument more and more, e.g. by ignoring upper / lower case differences and by ignoring the path of the DLL filename and file extension. If there is still no match at the end, the sampler will try to ignore the <module> argument completely and as a last resort search for an effect that only matches the given <effect-system> and <effect-name> arguments.

Possible Answers:

"OK[<effect-instance>]" -

in case the effect instance was successfully created, where <effect-instance> is the numerical ID of the new effect instance

"WRN:<warning-code>:<warning-message>" -

in case the effect instance was spawned successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

if the effect could not be instantiated

Examples:

C: "CREATE EFFECT_INSTANCE LADSPA '/usr/lib/ladspa/mod_delay_1419.so' 'modDelay'"

S: "OK[0]"

TOC

6.11.5. Creating an instance of an effect by its numerical ID

The front-end can spawn an instance of the desired effect by sending the following command:

CREATE EFFECT_INSTANCE <effect-index>

Where <effect-index> is the numerical ID of the effect as returned by the **"LIST AVAILABLE_EFFECTS"** command.

The sampler will try to load the requested effect and to create an instance of it.

Note: Since the numerical ID of a certain effect can change at any time, you should not use this command in LSCP files to restore a certain effect at a later time! To store a sampler session including all its effects, use the **portable text-based version of "CREATE EFFECT_INSTANCE"** instead! This allows to restore a sampler session with all its effects also on other machines, possibly even running a completely different operating system (e.g. Linux vs. Windows), with different plugin directories or plugin DLL names.

Possible Answers:

"OK[<effect-instance>]" -

in case the effect instance was successfully created, where <effect-instance> is the numerical ID of the new effect instance

"WRN:<warning-code>:<warning-message>" -

in case the effect instance was spawned successfully, but there are noteworthy issue(s) related, providing an appropriate warning code and warning message

"ERR:<error-code>:<error-message>" -

if the effect could not be instantiated

Examples:

C: "CREATE EFFECT_INSTANCE 72"

S: "OK[5]"

6.11.6. Destroy an effect instance

TOC

The front-end can destroy an unused effect instance and thus freeing it from memory by sending the following command:

DESTROY EFFECT_INSTANCE <effect-instance>

Where <effect-instance> is the numerical ID of the effect instance as returned by the **"CREATE EFFECT_INSTANCE"** or **"LIST EFFECT_INSTANCES"** command.

The effect instance can only be destroyed if it's not used in any part of the sampler's audio signal path anymore. If the effect instance is still in use somewhere, trying to destroy the effect instance will result in an error message.

Possible Answers:

"OK" -

in case the effect instance was successfully destroyed

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

C: "DESTROY EFFECT_INSTANCE 5"

S: "OK"

6.11.7. Retrieve amount of effect instances

TOC

The front-end can retrieve the current amount of effect instances by sending the following command:

GET EFFECT_INSTANCES

Possible Answers:

The sampler will answer by returning the current number of effect instances created and not yet destroyed in the current sampler session.

Examples:

C: "GET EFFECT_INSTANCES"

S: "14"

6.11.8. Get list of effect instances

TOC

The front-end can retrieve the current list of effect instances by sending the following command:

LIST EFFECT_INSTANCES

Possible Answers:

The sampler will answer by returning a comma separated list with numerical IDs of effects instances.

Example:

C: "LIST EFFECT_INSTANCES"

S: "9,11,14,15,16,17,25"

6.11.9. Retrieving current information about an effect instance

TOC

The front-end can ask for the current informations about a particular effect instance by sending the following command:

GET EFFECT_INSTANCE INFO <effect-instance>

Where <effect-instance> is the numerical ID of an effect instance as returned by the **"CREATE EFFECT_INSTANCE"** or **"LIST EFFECT_INSTANCES"** 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 information category. At the moment the following categories are defined:

SYSTEM -

name of the effect plugin system the effect is based on (e.g. "LADSPA")

MODULE -

module of the effect plugin system that contains this effect, the module is usually the dynamic-linked library (DLL) filename of the effect plugin, including full path (note that this filename may contain **escape sequences**)

NAME -

character string defining the unique name of the effect within its module (note that the character string may contain **escape sequences**)

DESCRIPTION -

human readable name of the effect, intended to be displayed in user interfaces (note that the character string may contain **escape sequences**)

INPUT_CONTROLS -

amount of input controls the effect instance provides, to allow controlling the effect parameters in realtime

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

Example:

```
C: "GET EFFECT_INSTANCE INFO 3"

S: "SYSTEM: LADSPA"

   "MODULE: /usr/lib/ladspa/mod_delay_1419.so"

   "NAME: modDelay"

   "DESCRIPTION: Modulatable delay"

   "INPUT_CONTROLS: 1"

   "."
```

TOC

6.11.10. Retrieving information about an effect parameter

Effects typically provide a certain set of effect parameters which can be altered by the user in realtime (e.g. depth of a reverb effect, duration of a delay effect, dry / wet signal ratio). Those controllable effect parameters are called "input controls". The front-end can ask for the current informations of an effect instance's input control by sending the following command:

```
GET EFFECT_INSTANCE_INPUT_CONTROL INFO <effect-instance> <input-control>
```

Where <effect-instance> is the numerical ID of an effect instance as returned by the **"CREATE EFFECT_INSTANCE"** or **"LIST EFFECT_INSTANCES"** command and <input-control> is the index of the input control within the numerical bounds as returned by the "INPUT_CONTROLS" field of the **"GET EFFECT_INSTANCE 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 information category. There are information categories which are always returned, independent of the respective effect parameter and there are optional information categories which are only shown for certain effect parameters. At the moment the following categories are defined:

DESCRIPTION -

(always returned) human readable name of the effect parameter, intended to be displayed in user interfaces (note that the character string may contain **escape sequences**)

VALUE -

(always returned) current (optional dotted) floating point value of this effect parameter

RANGE_MIN -

(optionally returned) minimum allowed value for this effect parameter

RANGE_MAX -

(optionally returned) maximum allowed value for this effect parameter

POSSIBILITIES -

(optionally returned) comma separated list of (optional dotted) floating point numbers, reflecting the exact set of possible values for this effect parameter

DEFAULT -

(optionally returned) default value of this effect parameter

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

Example:

```
C: "GET EFFECT_INSTANCE_INPUT_CONTROL INFO 1 0"

S: "DESCRIPTION: Base delay (s)"

   "VALUE: 0.500"

   "RANGE_MIN: 0.000"

   "."
```

6.11.11. Altering an effect parameter

TOC

The front-end can alter the current value of an effect parameter by sending the following command:

SET EFFECT_INSTANCE_INPUT_CONTROL VALUE <effect-instance> <input-control> <value>

Where <effect-instance> is the numerical ID of the effect instance as returned by the **"CREATE EFFECT_INSTANCE"** or **"LIST EFFECT_INSTANCES"** command, <input-control> is the index of the input control within the numerical bounds as returned by the "INPUT_CONTROLS" field of the **"GET EFFECT_INSTANCE INFO"** command and <value> is the new (optional dotted) floating point value for this effect parameter.

Possible Answers:

"OK" -

in case the effect was altered successfully

"ERR:<error-code>:<error-message>" -

in case it failed, providing an appropriate error code and error message

Examples:

```
C: "SET EFFECT_INSTANCE_INPUT_CONTROL VALUE 0 1 0.5"

S: "OK"
```

6.11.12. Retrieve amount of send effect chains

TOC

The front-end can retrieve the current amount of send effect chains of an audio output device by sending the following command:

```
GET SEND_EFFECT_CHAINS <audio-device>
```

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

Possible Answers:

> The sampler will answer by returning the current number of send effect chains of the supplied audio output device.

Examples:

```
C: "GET SEND_EFFECT_CHAINS 0"

S: "4"
```

6.11.13. Retrieve list of send effect chains

TOC

The front-end can retrieve the current list of send effect chains of an audio output device by sending the following command:

```
LIST SEND_EFFECT_CHAINS <audio-device>
```

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

Possible Answers:

> The sampler will answer by returning a comma separated list with numerical IDs of send effect chains of the supplied audio output device.

Examples:

```
C: "LIST SEND_EFFECT_CHAINS 0"

S: "3,4,7"
```

6.11.14. Add send effect chain

TOC

The front-end can add a send effect chain by sending the following command:

```
ADD SEND_EFFECT_CHAIN <audio-device>
```

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

Possible Answers:

"OK[<effect-chain>]" -

in case the send effect chain was added successfully, where <effect-chain> is the numerical ID of the new send effect chain

"ERR:<error-code>:<error-message>" -

if the send effect chain could not be added

Examples:

C: "ADD SEND_EFFECT_CHAIN 0"

S: "OK[2]"

6.11.15. Remove send effect chain

TOC

The front-end can remove a send effect chain by sending the following command:

REMOVE SEND_EFFECT_CHAIN <audio-device> <effect-chain>

Where <audio-device> should be replaced by the numerical ID of the audio output device as given by the **"CREATE AUDIO_OUTPUT_DEVICE"** or **"LIST AUDIO_OUTPUT_DEVICES"** command and <effect-chain> by the numerical ID as returned by the **"ADD SEND_EFFECT_CHAIN"** or **"LIST SEND_EFFECT_CHAINS"** command.

Possible Answers:

"OK" -

in case the send effect chain was removed successfully

"ERR:<error-code>:<error-message>" -

if the send effect chain could not be removed

Examples:

C: "REMOVE SEND_EFFECT_CHAIN 0 2"

S: "OK"

6.11.16. Retrieving information about a send effect chain

TOC

The front-end can ask for informations of a send effect chain by sending the following command:

GET SEND_EFFECT_CHAIN INFO <audio-device> <effect-chain>

Where <audio-device> should be replaced by the numerical ID of the audio output device as given by the **"CREATE AUDIO_OUTPUT_DEVICE"** or **"LIST AUDIO_OUTPUT_DEVICES"** command and <effect-chain> by the numerical ID as returned by the **"ADD SEND_EFFECT_CHAIN"** or **"LIST SEND_EFFECT_CHAINS"** 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 information category. At the moment the following categories are defined:

EFFECT_COUNT -

amount of effects in this send effect chain

EFFECT_SEQUENCE -

comma separated list of the numerical IDs of the effect instances in this send effect chain, in the order as they are procssed in the effect chain

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

Example:

C: "GET SEND_EFFECT_CHAIN INFO 0 2"

S: "EFFECT_COUNT: 3"

"EFFECT_SEQUENCE: 31,4,7"

"."

TOC

6.11.17. Append effect instance to a send effect chain

The front-end can add an unused effect instance to the end of a send effect chain by sending the following command:

APPEND SEND_EFFECT_CHAIN EFFECT <audio-device> <effect-chain> <effect-instance>

Where <audio-device> should be replaced by the numerical ID of the audio output device as given by the **"CREATE AUDIO_OUTPUT_DEVICE"** or **"LIST AUDIO_OUTPUT_DEVICES"** command and <effect-chain> by the numerical ID as returned by the **"ADD SEND_EFFECT_CHAIN"** or **"LIST SEND_EFFECT_CHAINS"** command and <effect-instance> as returned by the **"CREATE EFFECT_INSTANCE"** or **"LIST EFFECT_INSTANCES"** command.

Only unused effect instances can be added to the effect chain. Trying to add an effect instance which is already in use somewhere in the audio signal path of the sampler will result in an error.

Possible Answers:

"OK" -

in case the effect instance was added successfully to the chain

"ERR:<error-code>:<error-message>" -

if the effect instance could not be added

Examples:

C: "APPEND SEND_EFFECT_CHAIN EFFECT 0 2 38"

S: "OK"

6.11.18. Insert effect instance to a send effect chain

TOC

The front-end can add an unused effect instance to a certain position of a send effect chain by sending the following command:

> INSERT SEND_EFFECT_CHAIN EFFECT <audio-device> <effect-chain> <chain-pos> <effect-instance>

Where <audio-device> should be replaced by the numerical ID of the audio output device as given by the **"CREATE AUDIO_OUTPUT_DEVICE"** or **"LIST AUDIO_OUTPUT_DEVICES"** command, <effect-chain> by the numerical ID as returned by the **"ADD SEND_EFFECT_CHAIN"** or **"LIST SEND_EFFECT_CHAINS"** command, <effect-instance> as returned by the **"CREATE EFFECT_INSTANCE"** or **"LIST EFFECT_INSTANCES"** command and <chain-pos> the exact position of the effect chain where the supplied effect shall be inserted to.

Only unused effect instances can be added to the effect chain. Trying to add an effect instance which is already in use somewhere in the audio signal path of the sampler will result in an error.

Possible Answers:

> "OK" -
>
> > in case the effect instance was added successfully to the chain
>
> "ERR:<error-code>:<error-message>" -
>
> > if the effect instance could not be added

Examples:

> C: "INSERT SEND_EFFECT_CHAIN EFFECT 0 2 4 38"
>
> S: "OK"

6.11.19. Remove effect instance from send effect chain

TOC

The front-end can remove an effect instance from a certain position of a send effect chain by sending the following command:

> REMOVE SEND_EFFECT_CHAIN EFFECT <audio-device> <effect-chain> <chain-pos>

Where <audio-device> should be replaced by the numerical ID of the audio output device as given by the **"CREATE AUDIO_OUTPUT_DEVICE"** or **"LIST AUDIO_OUTPUT_DEVICES"** command, <effect-chain> by the numerical ID as returned by the **"ADD SEND_EFFECT_CHAIN"** or **"LIST SEND_EFFECT_CHAINS"** command and <chain-pos> the exact position of the effect instance to be removed from the effect chain.

Possible Answers:

> "OK" -
>
> > in case the effect instance was removed successfully
>
> "ERR:<error-code>:<error-message>" -

if the effect instance could not be removed

Examples:

C: "REMOVE SEND_EFFECT_CHAIN EFFECT 0 2 4"

S: "OK"

7. Command Syntax

TOC

The grammar of the control protocol as descibed in **Section 6** is defined below using Backus-Naur Form (BNF as described in **[RFC2234]**) where applicable.

input =

line

/ error

line =

statement LF

/ statement CR LF

statement =

/* epsilon (empty statement/line ignored) */

/ comment

/ command

comment =

'#'

/ comment '#'

/ comment SP

/ comment number

/ comment string

command =

ADD SP add_instruction

/ MAP SP map_instruction

/ UNMAP SP unmap_instruction

/ GET SP get_instruction

/ CREATE SP create_instruction

/ DESTROY SP destroy_instruction

/ LIST SP list_instruction

```
        / LOAD SP load_instruction

        / REMOVE SP remove_instruction

        / SET SP set_instruction

        / SUBSCRIBE SP subscribe_event

        / UNSUBSCRIBE SP unsubscribe_event

        / RESET SP reset_instruction

        / CLEAR SP clear_instruction

        / FIND SP find_instruction

        / MOVE SP move_instruction

        / COPY SP copy_instruction

        / EDIT SP edit_instruction

        / FORMAT SP format_instruction

        / SEND SP send_instruction

        / APPEND SP append_instruction

        / INSERT SP insert_instruction

        / RESET

        / QUIT

add_instruction =

        CHANNEL

        / CHANNEL SP MIDI_INPUT SP sampler_channel SP device_index

        / CHANNEL SP MIDI_INPUT SP sampler_channel SP device_index SP
        midi_input_port_index

        / DB_INSTRUMENT_DIRECTORY SP db_path

        / DB_INSTRUMENTS SP NON_MODAL SP scan_mode SP db_path SP filename

        / DB_INSTRUMENTS SP NON_MODAL SP scan_mode SP FILE_AS_DIR SP db_path
        SP filename

        / DB_INSTRUMENTS SP scan_mode SP db_path SP filename

        / DB_INSTRUMENTS SP scan_mode SP FILE_AS_DIR SP db_path SP filename

        / DB_INSTRUMENTS SP NON_MODAL SP db_path SP filename

        / DB_INSTRUMENTS SP NON_MODAL SP db_path SP filename SP instrument_index

        / DB_INSTRUMENTS SP db_path SP filename

        / DB_INSTRUMENTS SP db_path SP filename SP instrument_index

        / MIDI_INSTRUMENT_MAP
```

```
          / MIDI_INSTRUMENT_MAP SP map_name

          / SEND_EFFECT_CHAIN SP device_index

subscribe_event =

          AUDIO_OUTPUT_DEVICE_COUNT

          / AUDIO_OUTPUT_DEVICE_INFO

          / MIDI_INPUT_DEVICE_COUNT

          / MIDI_INPUT_DEVICE_INFO

          / CHANNEL_COUNT

          / CHANNEL_MIDI

          / DEVICE_MIDI

          / VOICE_COUNT

          / STREAM_COUNT

          / BUFFER_FILL

          / CHANNEL_INFO

          / FX_SEND_COUNT

          / FX_SEND_INFO

          / MIDI_INSTRUMENT_MAP_COUNT

          / MIDI_INSTRUMENT_MAP_INFO

          / MIDI_INSTRUMENT_COUNT

          / MIDI_INSTRUMENT_INFO

          / DB_INSTRUMENT_DIRECTORY_COUNT

          / DB_INSTRUMENT_DIRECTORY_INFO

          / DB_INSTRUMENT_COUNT

          / DB_INSTRUMENT_INFO

          / DB_INSTRUMENTS_JOB_INFO

          / MISCELLANEOUS

          / TOTAL_STREAM_COUNT

          / TOTAL_VOICE_COUNT

          / GLOBAL_INFO

          / EFFECT_INSTANCE_COUNT

          / EFFECT_INSTANCE_INFO

          / SEND_EFFECT_CHAIN_COUNT
          / SEND_EFFECT_CHAIN_INFO

unsubscribe_event =

          AUDIO_OUTPUT_DEVICE_COUNT

          / AUDIO_OUTPUT_DEVICE_INFO

          / MIDI_INPUT_DEVICE_COUNT

          / MIDI_INPUT_DEVICE_INFO

          / CHANNEL_COUNT

          / CHANNEL_MIDI

          / DEVICE_MIDI

          / VOICE_COUNT

          / STREAM_COUNT

          / BUFFER_FILL

          / CHANNEL_INFO

          / FX_SEND_COUNT

          / FX_SEND_INFO

          / MIDI_INSTRUMENT_MAP_COUNT

          / MIDI_INSTRUMENT_MAP_INFO

          / MIDI_INSTRUMENT_COUNT

          / MIDI_INSTRUMENT_INFO

          / DB_INSTRUMENT_DIRECTORY_COUNT

          / DB_INSTRUMENT_DIRECTORY_INFO

          / DB_INSTRUMENT_COUNT

          / DB_INSTRUMENT_INFO

          / DB_INSTRUMENTS_JOB_INFO

          / MISCELLANEOUS

          / TOTAL_STREAM_COUNT

          / TOTAL_VOICE_COUNT

          / GLOBAL_INFO

          / EFFECT_INSTANCE_COUNT

          / EFFECT_INSTANCE_INFO

          / SEND_EFFECT_CHAIN_COUNT

          / SEND_EFFECT_CHAIN_INFO
```

map_instruction =

MIDI_INSTRUMENT SP modal_arg midi_map SP midi_bank SP midi_prog SP engine_name SP filename SP instrument_index SP volume_value

/ MIDI_INSTRUMENT SP modal_arg midi_map SP midi_bank SP midi_prog SP engine_name SP filename SP instrument_index SP volume_value SP instr_load_mode

/ MIDI_INSTRUMENT SP modal_arg midi_map SP midi_bank SP midi_prog SP engine_name SP filename SP instrument_index SP volume_value SP entry_name

/ MIDI_INSTRUMENT SP modal_arg midi_map SP midi_bank SP midi_prog SP engine_name SP filename SP instrument_index SP volume_value SP instr_load_mode SP entry_name

unmap_instruction =

MIDI_INSTRUMENT SP midi_map SP midi_bank SP midi_prog

remove_instruction =

CHANNEL SP sampler_channel

/ CHANNEL SP MIDI_INPUT SP sampler_channel

/ CHANNEL SP MIDI_INPUT SP sampler_channel SP device_index

/ CHANNEL SP MIDI_INPUT SP sampler_channel SP device_index SP midi_input_port_index

/ MIDI_INSTRUMENT_MAP SP midi_map

/ MIDI_INSTRUMENT_MAP SP ALL

/ SEND_EFFECT_CHAIN SP device_index SP effect_chain

/ SEND_EFFECT_CHAIN SP EFFECT SP device_index SP effect_chain SP chain_pos

/ FX_SEND SP EFFECT SP sampler_channel SP fx_send_id

/ DB_INSTRUMENT_DIRECTORY SP FORCE SP db_path

/ DB_INSTRUMENT_DIRECTORY SP db_path

/ DB_INSTRUMENT SP db_path

get_instruction =

AVAILABLE_ENGINES

/ AVAILABLE_EFFECTS

/ EFFECT_INSTANCES

/ EFFECT SP INFO SP effect_index

/ EFFECT_INSTANCE SP INFO SP effect_instance

/ EFFECT_INSTANCE_INPUT_CONTROL SP INFO SP effect_instance SP input_control

/ SEND_EFFECT_CHAINS SP device_index

/ SEND_EFFECT_CHAIN SP INFO SP device_index SP effect_chain

/ AVAILABLE_MIDI_INPUT_DRIVERS

/ MIDI_INPUT_DRIVER SP INFO SP string

/ MIDI_INPUT_DRIVER_PARAMETER SP INFO SP string SP string

/ MIDI_INPUT_DRIVER_PARAMETER SP INFO SP string SP string SP key_val_list

/ AVAILABLE_AUDIO_OUTPUT_DRIVERS

/ AUDIO_OUTPUT_DRIVER SP INFO SP string

/ AUDIO_OUTPUT_DRIVER_PARAMETER SP INFO SP string SP string

/ AUDIO_OUTPUT_DRIVER_PARAMETER SP INFO SP string SP string SP key_val_list

/ AUDIO_OUTPUT_DEVICES

/ MIDI_INPUT_DEVICES

/ AUDIO_OUTPUT_DEVICE SP INFO SP number

/ MIDI_INPUT_DEVICE SP INFO SP number

/ MIDI_INPUT_PORT SP INFO SP number SP number

/ MIDI_INPUT_PORT_PARAMETER SP INFO SP number SP number SP string

/ AUDIO_OUTPUT_CHANNEL SP INFO SP number SP number

/ AUDIO_OUTPUT_CHANNEL_PARAMETER SP INFO SP number SP number SP string

/ 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

/ SERVER SP INFO

/ TOTAL_STREAM_COUNT

/ TOTAL_VOICE_COUNT

/ TOTAL_VOICE_COUNT_MAX

/ MIDI_INSTRUMENTS SP midi_map

/ MIDI_INSTRUMENTS SP ALL

/ MIDI_INSTRUMENT SP INFO SP midi_map SP midi_bank SP midi_prog

/ MIDI_INSTRUMENT_MAPS

```
/ MIDI_INSTRUMENT_MAP SP INFO SP midi_map

/ FX_SENDS SP sampler_channel

/ FX_SEND SP INFO SP sampler_channel SP fx_send_id

/ DB_INSTRUMENT_DIRECTORIES SP RECURSIVE SP db_path

/ DB_INSTRUMENT_DIRECTORIES SP db_path

/ DB_INSTRUMENT_DIRECTORY SP INFO SP db_path

/ DB_INSTRUMENTS SP RECURSIVE SP db_path

/ DB_INSTRUMENTS SP db_path

/ DB_INSTRUMENT SP INFO SP db_path

/ DB_INSTRUMENTS_JOB SP INFO SP number

/ VOLUME

/ VOICES

/ STREAMS

/ FILE SP INSTRUMENTS SP filename

/ FILE SP INSTRUMENT SP INFO SP filename SP instrument_index

set_instruction =

AUDIO_OUTPUT_DEVICE_PARAMETER SP number SP string '=' param_val_list

/ AUDIO_OUTPUT_CHANNEL_PARAMETER SP number SP number SP string '='
param_val_list

/ MIDI_INPUT_DEVICE_PARAMETER SP number SP string '=' param_val_list

/ MIDI_INPUT_PORT_PARAMETER SP number SP number SP string '=' NONE

/ MIDI_INPUT_PORT_PARAMETER SP number SP number SP string '=' param_val_list

/ EFFECT_INSTANCE_INPUT_CONTROL SP VALUE SP effect_instance SP
input_control SP control_value

/ CHANNEL SP set_chan_instruction

/ MIDI_INSTRUMENT_MAP SP NAME SP midi_map SP map_name

/ FX_SEND SP NAME SP sampler_channel SP fx_send_id SP fx_send_name

/ FX_SEND SP AUDIO_OUTPUT_CHANNEL SP sampler_channel SP fx_send_id SP
audio_channel_index SP audio_channel_index

/ FX_SEND SP MIDI_CONTROLLER SP sampler_channel SP fx_send_id SP midi_ctrl

/ FX_SEND SP LEVEL SP sampler_channel SP fx_send_id SP volume_value

/ FX_SEND SP EFFECT SP sampler_channel SP fx_send_id SP effect_chain SP
chain_pos
```

/ DB_INSTRUMENT_DIRECTORY SP NAME SP db_path SP stringval_escaped

/ DB_INSTRUMENT_DIRECTORY SP DESCRIPTION SP db_path SP stringval_escaped

/ DB_INSTRUMENT SP NAME SP db_path SP stringval_escaped

/ DB_INSTRUMENT SP DESCRIPTION SP db_path SP stringval_escaped

/ DB_INSTRUMENT SP FILE_PATH SP filename SP filename

/ ECHO SP boolean

/ SHELL SP INTERACT SP boolean

/ SHELL SP AUTO_CORRECT SP boolean

/ SHELL SP DOC SP boolean

/ VOLUME SP volume_value

/ VOICES SP number

/ STREAMS SP number

create_instruction =

AUDIO_OUTPUT_DEVICE SP string SP key_val_list

/ AUDIO_OUTPUT_DEVICE SP string

/ MIDI_INPUT_DEVICE SP string SP key_val_list

/ MIDI_INPUT_DEVICE SP string

/ FX_SEND SP sampler_channel SP midi_ctrl

/ FX_SEND SP sampler_channel SP midi_ctrl SP fx_send_name

/ EFFECT_INSTANCE SP effect_index

/ EFFECT_INSTANCE SP effect_system SP module SP effect_name

reset_instruction =

CHANNEL SP sampler_channel

clear_instruction =

MIDI_INSTRUMENTS SP midi_map

/ MIDI_INSTRUMENTS SP ALL

find_instruction =

DB_INSTRUMENTS SP NON_RECURSIVE SP db_path SP query_val_list

/ DB_INSTRUMENTS SP db_path SP query_val_list

/ DB_INSTRUMENT_DIRECTORIES SP NON_RECURSIVE SP db_path SP query_val_list

/ DB_INSTRUMENT_DIRECTORIES SP db_path SP query_val_list

```
        / LOST SP DB_INSTRUMENT_FILES

move_instruction =

        DB_INSTRUMENT_DIRECTORY SP db_path SP db_path

        / DB_INSTRUMENT SP db_path SP db_path

copy_instruction =

        DB_INSTRUMENT_DIRECTORY SP db_path SP db_path

        / DB_INSTRUMENT SP db_path SP db_path

destroy_instruction =

        AUDIO_OUTPUT_DEVICE SP number

        / MIDI_INPUT_DEVICE SP number

        / FX_SEND SP sampler_channel SP fx_send_id

        / EFFECT_INSTANCE SP number

load_instruction =

        INSTRUMENT SP load_instr_args

        / ENGINE SP load_engine_args

append_instruction =

        SEND_EFFECT_CHAIN SP EFFECT SP device_index SP effect_chain SP
        effect_instance

insert_instruction =

        SEND_EFFECT_CHAIN SP EFFECT SP device_index SP effect_chain SP chain_pos
        SP effect_instance

set_chan_instruction =

        AUDIO_OUTPUT_DEVICE SP sampler_channel SP device_index

        / AUDIO_OUTPUT_CHANNEL SP sampler_channel SP audio_channel_index SP
        audio_channel_index

        / AUDIO_OUTPUT_TYPE SP sampler_channel SP audio_output_type_name

        / MIDI_INPUT SP sampler_channel SP device_index SP midi_input_port_index SP
        midi_input_channel_index

        / MIDI_INPUT_DEVICE SP sampler_channel SP device_index

        / MIDI_INPUT_PORT SP sampler_channel SP midi_input_port_index

        / MIDI_INPUT_CHANNEL SP sampler_channel SP midi_input_channel_index

        / MIDI_INPUT_TYPE SP sampler_channel SP midi_input_type_name

        / VOLUME SP sampler_channel SP volume_value
```

```
          / MUTE SP sampler_channel SP boolean

          / SOLO SP sampler_channel SP boolean

          / MIDI_INSTRUMENT_MAP SP sampler_channel SP midi_map

          / MIDI_INSTRUMENT_MAP SP sampler_channel SP NONE

          / MIDI_INSTRUMENT_MAP SP sampler_channel SP DEFAULT

edit_instruction =

          CHANNEL SP INSTRUMENT SP sampler_channel

format_instruction =

          INSTRUMENTS_DB

modal_arg =

          /* epsilon (empty argument) */

          / NON_MODAL SP

key_val_list =

          string '=' param_val_list

          / key_val_list SP string '=' param_val_list

buffer_size_type =

          BYTES

          / PERCENTAGE

list_instruction =

          AUDIO_OUTPUT_DEVICES

          / MIDI_INPUT_DEVICES

          / CHANNELS

          / CHANNEL SP MIDI_INPUTS SP sampler_channel

          / AVAILABLE_ENGINES

          / AVAILABLE_EFFECTS

          / EFFECT_INSTANCES

          / SEND_EFFECT_CHAINS SP number

          / AVAILABLE_MIDI_INPUT_DRIVERS

          / AVAILABLE_AUDIO_OUTPUT_DRIVERS

          / MIDI_INSTRUMENTS SP midi_map

          / MIDI_INSTRUMENTS SP ALL

          / MIDI_INSTRUMENT_MAPS
```

```
        / FX_SENDS SP sampler_channel

        / DB_INSTRUMENT_DIRECTORIES SP RECURSIVE SP db_path

        / DB_INSTRUMENT_DIRECTORIES SP db_path

        / DB_INSTRUMENTS SP RECURSIVE SP db_path

        / DB_INSTRUMENTS SP db_path

        / FILE SP INSTRUMENTS SP filename

send_instruction =

        CHANNEL SP MIDI_DATA SP string SP sampler_channel SP number SP number

load_instr_args =

        filename SP instrument_index SP sampler_channel

        / NON_MODAL SP filename SP instrument_index SP sampler_channel

load_engine_args =

        engine_name SP sampler_channel

instr_load_mode =

        ON_DEMAND

        / ON_DEMAND_HOLD

        / PERSISTENT

effect_instance =

        number

device_index =

        number

audio_channel_index =

        number

audio_output_type_name =

        string

midi_input_port_index =

        number

midi_input_channel_index =

        number

        / ALL

midi_input_type_name =

        string
```

```
midi_map =
          number
midi_bank =
          number
midi_prog =
          number
midi_ctrl =
          number
volume_value =
          dotnum
          / number
control_value =
          real
sampler_channel =
          number
instrument_index =
          number
fx_send_id =
          number
engine_name =
          string
filename =
          path
db_path =
          path
map_name =
          stringval_escaped
entry_name =
          stringval_escaped
fx_send_name =
          stringval_escaped
effect_name =
    stringval_escaped
effect_index =
    number
effect_chain =
    number
chain_pos =
    number
input_control =
    number
param_val_list =
    param_val
    / param_val_list','param_val
param_val =
    string
    / stringval
    / number
    / dotnum
query_val_list =
    string '=' query_val
    / query_val_list SP string '=' query_val
query_val =
    text_escaped
    / stringval_escaped
scan_mode =
    RECURSIVE
    / NON_RECURSIVE
    / FLAT
effect_system =
    string
module =
    filename
```

7.1. Character Set and Escape Sequences

Older versions of this protocol up to and including v1.1 only supported the standard ASCII character set (ASCII code 0 - 127) **[RFC20]**, all younger versions of this protocol however support the Extended ASCII character set (ASCII code 0 - 255). The same group of younger protocols also support escape sequences, but only for certain, explicitly declared parts of the protocol. The supported escape sequences are defined as follows:

ASCII Character Sequence	Translated into (Name)
\n	new line
\r	carriage return
\f	form feed
\t	horizontal tab
\v	vertical tab
\'	apostrophe
\"	quotation mark
\\	backslash
\OOO	three digit octal ASCII code of the character
\xHH	two digit hex ASCII code of the character

Notice: due to the transition of certain parts of the protocol which now support escape sequences, a slight backward incompatibility to protocols version v1.1 and younger has been introduced. The only difference is that in parts of the protocol where escape characters are now supported, a backslash characters MUST be escaped as well (that is as double backslash), whereas in the old versions a single backslash was sufficient.

The following LSCP commands support escape sequences as part of their filename / path based arguments and / or may contain a filename / path with escape sequences in their response:

"LOAD INSTRUMENT"

"GET CHANNEL INFO"

"MAP MIDI_INSTRUMENT"

"GET MIDI_INSTRUMENT INFO"

"ADD DB_INSTRUMENT_DIRECTORY"

"ADD DB_INSTRUMENTS"

"REMOVE DB_INSTRUMENT_DIRECTORY"

"REMOVE DB_INSTRUMENT"

"GET DB_INSTRUMENT_DIRECTORIES"

"LIST DB_INSTRUMENT_DIRECTORIES"

"GET DB_INSTRUMENT_DIRECTORY INFO"

"GET DB_INSTRUMENTS"

"LIST DB_INSTRUMENTS"

"GET DB_INSTRUMENT INFO"

"SET DB_INSTRUMENT_DIRECTORY NAME"

"SET DB_INSTRUMENT_DIRECTORY DESCRIPTION"

"SET DB_INSTRUMENT NAME"

"SET DB_INSTRUMENT DESCRIPTION"

"FIND DB_INSTRUMENTS"

"FIND DB_INSTRUMENT_DIRECTORIES"

"MOVE DB_INSTRUMENT"

"MOVE DB_INSTRUMENT_DIRECTORY"

"COPY DB_INSTRUMENT"

"COPY DB_INSTRUMENT_DIRECTORY"

"FIND LOST DB_INSTRUMENT_FILES"

"SET DB_INSTRUMENT FILE_PATH"

"GET FILE INSTRUMENTS"

"LIST FILE INSTRUMENTS"

"GET FILE INSTRUMENT INFO"

"GET EFFECT INFO"

"GET EFFECT_INSTANCE INFO"

"CREATE EFFECT_INSTANCE"

Note that the forward slash character ('/') has a special meaning in filename / path based arguments: it acts as separator of the nodes in the path, thus if a directory- or filename includes a forward slash (not intended as path node separator), you MUST escape that slash either with the respective hex escape sequence ("\x2f") or with the respective octal escape sequence ("\057").

Note for Windows: file path arguments in LSCP are expected to use forward slashes as directory node separator similar to Unix based operating systems. In contrast to Unix however a Windows typical drive character is expected to be prefixed to the path. That is an original Windows file path like "D:\Sounds\My.gig" would become in LSCP: "D:/Sounds/My.gig".

The following LSCP commands even support escape sequences as part of at least one of their text-based arguments (i.e. entity name, description) and / or may contain escape sequences in at least one of their text-based fields in their response:

"GET SERVER INFO"

"GET ENGINE INFO"

"GET CHANNEL INFO"

"CREATE FX_SEND"

"GET FX_SEND INFO"

"SET FX_SEND NAME"

"GET MIDI_INSTRUMENT INFO"

"GET MIDI_INSTRUMENT_MAP INFO"

"ADD MIDI_INSTRUMENT_MAP"

"MAP MIDI_INSTRUMENT"

"SET MIDI_INSTRUMENT_MAP NAME"

"GET DB_INSTRUMENT_DIRECTORY INFO"

"SET DB_INSTRUMENT_DIRECTORY NAME"

"SET DB_INSTRUMENT_DIRECTORY DESCRIPTION"

"FIND DB_INSTRUMENT_DIRECTORIES"

"GET DB_INSTRUMENT INFO"

"SET DB_INSTRUMENT NAME"

"SET DB_INSTRUMENT DESCRIPTION"

"FIND DB_INSTRUMENTS"

"GET EFFECT INFO"

"GET EFFECT_INSTANCE INFO"

"CREATE EFFECT_INSTANCE"

Please note that these lists are manually maintained. If you find a command that also supports escape sequences we forgot to mention here, please report it!

TOC

8. Events

This chapter will describe all currently defined events supported by LinuxSampler.

TOC

8.1. Number of audio output devices changed

Client may want to be notified when the total number of audio output devices on the back-end changes by issuing the following command:

```
SUBSCRIBE AUDIO_OUTPUT_DEVICE_COUNT
```

Server will start sending the following notification messages:

```
"NOTIFY:AUDIO_OUTPUT_DEVICE_COUNT:<devices>"
```

where <devices> will be replaced by the new number of audio output devices.

TOC

8.2. Audio output device's settings changed

Client may want to be notified when changes were made to audio output devices on the back-end by issuing the following command:

```
SUBSCRIBE AUDIO_OUTPUT_DEVICE_INFO
```

Server will start sending the following notification messages:

```
"NOTIFY:AUDIO_OUTPUT_DEVICE_INFO:<device-id>"
```

where <device-id> will be replaced by the numerical ID of the audio output device, which settings has been changed. The front-end will have to send the respective command to actually get the audio output device info. Because these messages will be triggered by LSCP commands issued by other clients rather than real time events happening on the server, it is believed that an empty notification message is sufficient here.

TOC

8.3. Number of MIDI input devices changed

Client may want to be notified when the total number of MIDI input devices on the back-end changes by issuing the following command:

```
SUBSCRIBE MIDI_INPUT_DEVICE_COUNT
```

Server will start sending the following notification messages:

```
"NOTIFY:MIDI_INPUT_DEVICE_COUNT:<devices>"
```

where <devices> will be replaced by the new number of MIDI input devices.

TOC

8.4. MIDI input device's settings changed

Client may want to be notified when changes were made to MIDI input devices on the back-end by issuing the following command:

```
SUBSCRIBE MIDI_INPUT_DEVICE_INFO
```

Server will start sending the following notification messages:

```
"NOTIFY:MIDI_INPUT_DEVICE_INFO:<device-id>"
```

where <device-id> will be replaced by the numerical ID of the MIDI input device, which settings has been changed. The front-end will have to send the respective command to actually get the MIDI input device info. Because these messages will be triggered by LSCP commands issued by other clients rather than real time events happening on the server, it is believed that an empty notification message is sufficient here.

TOC

8.5. Number of sampler channels changed

Client may want to be notified when the total number of channels on the back-end changes by issuing the following command:

```
SUBSCRIBE CHANNEL_COUNT
```

Server will start sending the following notification messages:

```
"NOTIFY:CHANNEL_COUNT:<channels>"
```

where <channels> will be replaced by the new number of sampler channels.

8.6. MIDI data on a sampler channel arrived

Client may want to be notified when MIDI data arrive on sampler channels on back-end side, by issuing the following command:

```
SUBSCRIBE CHANNEL_MIDI
```

Server will start sending one of the the following notification messages:

```
"NOTIFY:CHANNEL_MIDI:<channel-id> NOTE_ON <note> <velocity>"

"NOTIFY:CHANNEL_MIDI:<channel-id> NOTE_OFF <note> <velocity>"
```

where <channel-id> will be replaced by the ID of the sampler channel where the MIDI data arrived. <note> and <velocity> are integer values in the range between 0 .. 127, reflecting the analog meaning of the MIDI specification.

CAUTION: no guarantee whatsoever will be made that MIDI events are actually all delivered by this mechanism! With other words: events could be lost at any time! This restriction was made to keep the RT-safeness of the backend's MIDI and audio thread unaffected by this feature.

8.7. MIDI data on a MIDI input device arrived

Client may want to be notified when MIDI data arrive on MIDI input devices by issuing the following command:

```
SUBSCRIBE DEVICE_MIDI
```

Server will start sending one of the the following notification messages:

```
"NOTIFY:DEVICE_MIDI:<device-id> <port-id> NOTE_ON <note> <velocity>"

"NOTIFY:DEVICE_MIDI:<device-id> <port-id> NOTE_OFF <note> <velocity>"
```

where <device-id> <port-id> will be replaced by the IDs of the respective MIDI input device and the device's MIDI port where the MIDI data arrived. <note> and <velocity> are integer values in the range between 0 .. 127, reflecting the analog meaning of the MIDI specification.

CAUTION: no guarantee whatsoever will be made that MIDI events are actually all delivered by this mechanism! With other words: events could be lost at any time! This restriction was made to keep the RT-safeness of the backend's MIDI and audio thread unaffected by this feature.

8.8. Number of active voices changed

Client may want to be notified when the number of voices on the back-end changes by issuing the following command:

```
SUBSCRIBE VOICE_COUNT
```

Server will start sending the following notification messages:

```
"NOTIFY: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.

8.9. Number of active disk streams changed

TOC

Client may want to be notified when the number of streams on the back-end changes by issuing the following command: SUBSCRIBE STREAM_COUNT

```
SUBSCRIBE STREAM_COUNT
```

Server will start sending the following notification messages:

```
"NOTIFY:STREAM_COUNT:<sampler-channel> <streams>"
```

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

8.10. Disk stream buffer fill state changed

TOC

Client may want to be notified when the buffer fill state of a disk stream on the back-end changes by issuing the following command:

```
SUBSCRIBE BUFFER_FILL
```

Server will start sending the following notification messages:

```
"NOTIFY:BUFFER_FILL:<sampler-channel> <fill-data>"
```

where <sampler-channel> will be replaced by the sampler channel the buffer fill state change occurred on and <fill-data> will be replaced by the buffer fill data for this channel as described in **Section 6.4.13** as if the **"GET CHANNEL BUFFER_FILL PERCENTAGE"** command was issued on this channel.

8.11. Channel information changed

TOC

Client may want to be notified when changes were made to sampler channels on the back-end by issuing the following command:

```
SUBSCRIBE CHANNEL_INFO
```

Server will start sending the following notification messages:

```
"NOTIFY:CHANNEL_INFO:<sampler-channel>"
```

where <sampler-channel> will be replaced by the sampler channel the channel info change occurred.

The front-end will have to send the respective command to actually get the channel info. Because these messages will be triggered by LSCP commands issued by other clients rather than real time events happening on the server, it is believed that an empty notification message is sufficient here.

8.12. Number of effect sends changed

TOC

Client may want to be notified when the number of effect sends on a particular sampler channel is changed by issuing the following command:

```
SUBSCRIBE FX_SEND_COUNT
```

Server will start sending the following notification messages:

```
"NOTIFY:FX_SEND_COUNT:<channel-id> <fx-sends>"
```

where <channel-id> will be replaced by the numerical ID of the sampler channel, on which the effect sends number is changed and <fx-sends> will be replaced by the new number of effect sends on that channel.

8.13. Effect send information changed

TOC

Client may want to be notified when changes were made to effect sends on a a particular sampler channel by issuing the following command:

```
SUBSCRIBE FX_SEND_INFO
```

Server will start sending the following notification messages:

```
"NOTIFY:FX_SEND_INFO:<channel-id> <fx-send-id>"
```

where <channel-id> will be replaced by the numerical ID of the sampler channel, on which an effect send entity is changed and <fx-send-id> will be replaced by the numerical ID of the changed effect send.

8.14. Total number of active voices changed

TOC

Client may want to be notified when the total number of voices on the back-end changes by issuing the following command:

```
SUBSCRIBE TOTAL_VOICE_COUNT
```

Server will start sending the following notification messages:

```
"NOTIFY:TOTAL_VOICE_COUNT:<voices>"
```

where <voices> will be replaced by the new number of all currently active voices.

8.15. Total number of active disk streams changed

TOC

Client may want to be notified when the total number of disk streams on the back-end changes by issuing the following command:

```
SUBSCRIBE TOTAL_STREAM_COUNT
```

Server will start sending the following notification messages:

```
"NOTIFY:TOTAL_STREAM_COUNT:<streams>"
```

where <streams> will be replaced by the new number of all currently active disk streams.

8.16. Number of MIDI instrument maps changed

TOC

Client may want to be notified when the number of MIDI instrument maps on the back-end changes by issuing the following command:

```
SUBSCRIBE MIDI_INSTRUMENT_MAP_COUNT
```

Server will start sending the following notification messages:

```
"NOTIFY:MIDI_INSTRUMENT_MAP_COUNT:<maps>"
```

where <maps> will be replaced by the new number of MIDI instrument maps.

8.17. MIDI instrument map information changed

TOC

Client may want to be notified when changes were made to MIDI instrument maps on the back-end by issuing the following command:

```
SUBSCRIBE MIDI_INSTRUMENT_MAP_INFO
```

Server will start sending the following notification messages:

```
"NOTIFY:MIDI_INSTRUMENT_MAP_INFO:<map-id>"
```

where <map-id> will be replaced by the numerical ID of the MIDI instrument map, for which information changes occurred. The front-end will have to send the respective command to actually get the MIDI instrument map info. Because these messages will be triggered by LSCP commands issued by other clients rather than real time events happening on the server, it is believed that an empty notification message is sufficient here.

8.18. Number of MIDI instruments changed

TOC

Client may want to be notified when the number of MIDI instrument maps on the back-end changes by issuing the following command:

```
SUBSCRIBE MIDI_INSTRUMENT_COUNT
```

Server will start sending the following notification messages:

```
"NOTIFY:MIDI_INSTRUMENT_COUNT:<map-id> <instruments>"
```

where <map-id> is the numerical ID of the MIDI instrument map, in which the nuber of instruments has changed and <instruments> will be replaced by the new number of MIDI instruments in the specified map.

8.19. MIDI instrument information changed

TOC

Client may want to be notified when changes were made to MIDI instruments on the back-end by issuing the following command:

```
SUBSCRIBE MIDI_INSTRUMENT_INFO
```

Server will start sending the following notification messages:

```
"NOTIFY:MIDI_INSTRUMENT_INFO:<map-id> <bank> <program>"
```

where <map-id> will be replaced by the numerical ID of the MIDI instrument map, in which a MIDI instrument is changed. <bank> and <program> specifies the location of the changed MIDI instrument in the map. The front-end will have to send the respective command to actually get the MIDI instrument info. Because these messages will be triggered by LSCP commands issued by other clients rather than real time events happening on the server, it is believed that an empty notification message is sufficient here.

8.20. Global settings changed

TOC

Client may want to be notified when changes to the global settings of the sampler were made by issuing the following command:

```
SUBSCRIBE GLOBAL_INFO
```

Server will start sending the following types of notification messages:

"NOTIFY:GLOBAL_INFO:VOLUME <volume>" - Notifies that the golbal volume of the sampler is changed, where <volume> will be replaced by the optional dotted floating point value, reflecting the new global volume parameter.

"NOTIFY:GLOBAL_INFO:VOICES <max-voices>" - Notifies that the golbal limit of the sampler for maximum voices is changed, where <max-voices> will be an integer value, reflecting the new global voice limit parameter.

"NOTIFY:GLOBAL_INFO:STREAMS <max-streams>" - Notifies that the golbal limit of the sampler for maximum disk streams is changed, where <max-streams> will be an integer value, reflecting the new global disk streams limit parameter.

8.21. Number of database instrument directories changed

TOC

Client may want to be notified when the number of instrument directories in a particular directory in the instruments database is changed by issuing the following command:

```
SUBSCRIBE DB_INSTRUMENT_DIRECTORY_COUNT
```

Server will start sending the following notification messages:

```
"NOTIFY:DB_INSTRUMENT_DIRECTORY_COUNT:<dir-path>"
```

where <dir-path> will be replaced by the absolute path name of the directory in the instruments database, in which the number of directories is changed.

Note that when a non-empty directory is removed, this event is not sent for the subdirectories in that directory.

TOC

8.22. Database instrument directory information changed

Client may want to be notified when changes were made to directories in the instruments database by issuing the following command:

```
SUBSCRIBE DB_INSTRUMENT_DIRECTORY_INFO
```

Server will start sending the following notification messages:

```
"NOTIFY:DB_INSTRUMENT_DIRECTORY_INFO:<dir-path>"
```

where <dir-path> will be replaced by the absolute path name of the directory, for which information changes occurred. The front-end will have to send the respective command to actually get the updated directory info. Because these messages will be triggered by LSCP commands issued by other clients rather than real time events happening on the server, it is believed that an empty notification message is sufficient here.

```
"NOTIFY:DB_INSTRUMENT_DIRECTORY_INFO:NAME <old-dir-path> <new-name>"
```

where <old-dir-path> is the old absolute path name of the directory (encapsulated into apostrophes), which name is changes and <new-name> is the new name of the directory, encapsulated into apostrophes.

TOC

8.23. Number of database instruments changed

Client may want to be notified when the number of instruments in a particular directory in the instruments database is changed by issuing the following command:

```
SUBSCRIBE DB_INSTRUMENT_COUNT
```

Server will start sending the following notification messages:

```
"NOTIFY:DB_INSTRUMENT_COUNT:<dir-path>"
```

where <dir-path> will be replaced by the absolute path name of the directory in the instruments database, in which the number of instruments is changed.

Note that when a non-empty directory is removed, this event is not sent for the instruments in that directory.

TOC

8.24. Database instrument information changed

Client may want to be notified when changes were made to instruments in the instruments database by issuing the following command:

```
SUBSCRIBE DB_INSTRUMENT_INFO
```

Server will start sending the following notification messages:

"NOTIFY:DB_INSTRUMENT_INFO:<instr-path>"

where <instr-path> will be replaced by the absolute path name of the instrument, which settings are changed. The front-end will have to send the respective command to actually get the updated directory info. Because these messages will be triggered by LSCP commands issued by other clients rather than real time events happening on the server, it is believed that an empty notification message is sufficient here.

"NOTIFY:DB_INSTRUMENT_INFO:NAME <old-instr-path> <new-name>"

where <old-instr-path> is the old absolute path name of the instrument (encapsulated into apostrophes), which name is changes and <new-name> is the new name of the instrument, encapsulated into apostrophes.

8.25. Database job status information changed

TOC

Client may want to be notified when the status of particular database instruments job is changed by issuing the following command:

SUBSCRIBE DB_INSTRUMENTS_JOB_INFO

Server will start sending the following notification messages:

"NOTIFY:DB_INSTRUMENTS_JOB_INFO:<job-id>"

where <job-id> will be replaced by the numerical ID of the job, which status is changed. The front-end will have to send the respective command to actually get the status info. Because these messages will be triggered by LSCP commands issued by other clients rather than real time events happening on the server, it is believed that an empty notification message is sufficient here.

8.26. Number of effect instances changed

TOC

Client may want to be notified when the number of effect instances is changed by issuing the following command:

SUBSCRIBE EFFECT_INSTANCE_COUNT

Server will start sending the following notification messages:

"EFFECT_INSTANCE_COUNT:<instances>"

where <instances> will be replaced by the new number of effect instances.

8.27. Effect instance information changed

TOC

Client may want to be notified when changes were made to effect instances on the back-end by issuing the following command:

SUBSCRIBE EFFECT_INSTANCE_INFO

Server will start sending the following notification messages:

> "EFFECT_INSTANCE_INFO:<instance-id>"

where <instance-id> will be replaced by the numerical ID of the effect instance.

8.28. Number of send effect chains changed

TOC

Client may want to be notified when the number of send effect chains is changed by issuing the following command:

> SUBSCRIBE SEND_EFFECT_CHAIN_COUNT

Server will start sending the following notification messages:

> "NOTIFY:SEND_EFFECT_CHAIN_COUNT:<device-id> <chains>"

where <device-id> will be replaced by the numerical ID of the audio output device, in which the number of send effect chains is changed and <chains> will be replaced by the new number of send effect chains.

8.29. Send effect chain information changed

TOC

Client may want to be notified when changes were made to send effect chains on the back-end by issuing the following command:

> SUBSCRIBE SEND_EFFECT_CHAIN_INFO

Server will start sending the following notification messages:

> "SEND_EFFECT_CHAIN_INFO:<device-id> <chain-id> <instances>" - Notifies that the number of effect instances in a particular send effect chain is changed, where <device-id> will be replaced by the numerical ID of the audio output device the send effect chain belongs to, <chain-id> will be replaced by the numerical ID of the send effect chain in which the number of effect instances has changed and <instances> will be replaced by the new number of effect instances in the specified send effect chain.

8.30. Miscellaneous and debugging events

TOC

Client may want to be notified of miscellaneous and debugging events occurring at the server by issuing the following command:

> SUBSCRIBE MISCELLANEOUS

Server will start sending the following notification messages:

> "NOTIFY:MISCELLANEOUS:<string>"

where <string> will be replaced by whatever data server wants to send to the client. Client MAY display this data to the user AS IS to facilitate debugging.

---

9. Security Considerations

TOC

As there is so far no method of authentication and authorization 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.

TOC

10. Acknowledgments

This document has benefited greatly from the comments of the following people, discussed on the LinuxSampler developer's mailing list:

Rui Nuno Capela

Vladimir Senkov

Mark Knecht

Grigor Iliev

TOC

11. References

[RFC20]	UCLA, "ASCII format for Network Interchange," RFC 20, 1969.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119, 1997.
[RFC2234]	Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications," RFC 2234, 1997.
[RFC793]	Defense Advanced Research Projects Agency, "TRANSMISSION CONTROL PROTOCOL," RFC 793, 1981.

TOC

Author's Address

	C. Schoenebeck
	LinuxSampler.org
	Crudebyte Engineering
	Hofgartenstr. 3
	74189 Weinsberg
	Germany
Phone:	+49 7134 911614
Email:	cuse@users.sf.net

TOC

Full Copyright Statement

Intellectual Property