PCM (digital audio) plugins

PCM plugins provide functionality and features of PCM devices. The plugins take care about various sample conversions, sample copying among channels and so on.

Slave definition

The slave plugin can be specified directly with a string or the definition can be entered inside a compound configuration node. Some restrictions can be also specified (the static rate or constant period time).

```
pcm.slave_name {
        type pcm              # PCM name
        slave {               # PCM definition
                pcm "hw:0,0"  # or pcm "plughw:0,0"
                rate 44100    # or rate 44100!or rate[:rate]*
                period_time 0 # or period_time 0!or period_time[:time]*
                period_size 0 # or period_size 0!or period_size[:size]*
                buffer_size 0 # or buffer_size 0!or buffer_size[:size]*
        }
}
```

Example:

```
pcm.slave_slave_capturedmix {
        type dmix
        slave.pcm "hw:0,0"
}

pcm.slave_dmix {
        type dmix
        slave {
                pcm "hw:0,0"
                period_time 0
                buffer_size 65536
        }
}
```

The equivalent configuration (in one composed):

```
pcm.slave_dmix {
        type dmix
        slave {
                pcm {
                        type hw
                        card 0
                        device 0
                }
                period_time 0
                buffer_size 65536
        }
}
```

Plugin: hw

This plugin communication directly with the ALSA kernel driver. It is a raw communication without any conversion. The emulation of mmap access can be optionally enabled, but unpaired mmap access is not allowed.

The control actions specified for the device to operate in a non-blocking manner. Note that the blocking behavior of the read/write/mmap access can be changed by this option. This influences the behavior of the PCM API functions. The PCM API functions with the timeout argument behave in accordance with the value kept with this option.

```
pcm.name {
        type hw               # Direct PCM
        card 0                # Card name or number (integer)
        device 0              # Device number (integer)
        subdevice -1          # Subdevice number (-1 for any)
        nonblock yes[:1] no[:0] # Control the blocking behavior for control ioctlcmds
        sync_ptr_ioctl yes[:1] no[:0] # Use sync_ptr ioctl
        mmap_rw yes[:1] no[:0] # Use mmap_rw for data transfer
        disable_mmap yes[:1] no[:0] # Disallow mmap for data access
        disable_mlock yes[:1] no[:0] # Disallow mlock for mmap buffer
        force_rate yes[:1] no[:0] # Force the hardware rate
        force_period_time yes[:1] no[:0] # Force the hardware period time
        force_buffer_size yes[:1] no[:0] # Force the hardware buffer size
        sync_ptr_ioctl_period_time yes[:1] no[:0] # Use sync_ptr ioctl for period updating
}
```

Function reference

- snd_pcm_hw_open()
- snd_pcm_hw_open()

Plugin: mmap_emul

```
pcm.name {
        type mmap_emul
        slave PCM
}
```

Function reference

- snd_pcm_hw_open()

Plugin: null

This plugin provides controls of a PCM stream or creates a stream with zero samples.

Note: This implementation uses devices (kernel jiffies/timers) must be enabled, and /dev/null (/nullzero) must be readable.

```
pcm.name {
        type null             # Null PCM
        time yes[:1] no[:0]   # Project channel empty PWP (e.g.) silence
}
```

Function reference

- snd_pcm_null_open()
- snd_pcm_null_open()

Plugin: copy

This plugin copies samples from master copy PCM to given slave PCM. The channel count, format and rate must match for both of them.

```
pcm.name {
        type copy             # Copy PCM
        slave PCM             # Slave PCM
        slave {               # PCM definition
                pcm "hw:0,0"  # or pcm "plughw:0,0"
                rate 44100    # or rate 44100!or rate[:rate]*
                period_time 0 # or period_time 0!or period_time[:time]*
                period_size 0 # or period_size 0!or period_size[:size]*
                buffer_size 0 # or buffer_size 0!or buffer_size[:size]*
        }
}
```

Function reference

- snd_pcm_copy_open()
- snd_pcm_copy_open()

Plugin: linear

This plugin converts linear samples from master linear conversion PCM to given slave PCM. The channel count, format and rate must match for both of them.

```
pcm.name {
        type linear           # Linear conversion PCM
        slave PCM             # Slave PCM
        slave {               # PCM definition
                pcm "hw:0,0"  # or pcm "plughw:0,0"
                rate 44100    # or rate 44100!or rate[:rate]*
                period_time 0 # or period_time 0!or period_time[:time]*
                period_size 0 # or period_size 0!or period_size[:size]*
                buffer_size 0 # or buffer_size 0!or buffer_size[:size]*
        }
}
```

Function reference

- snd_pcm_linear_open()
- snd_pcm_linear_open()

Plugin: linear-r-float

This plugin converts linear samples from float samples from master linear-r-float conversion PCM to given slave PCM. The channel count, format and rate must match for both of them.

```
pcm.name {
        type linear-r-float   # Linear-r-float conversion PCM
        slave PCM             # Slave PCM
        slave {               # PCM definition
                pcm "hw:0,0"  # or pcm "plughw:0,0"
                rate 44100    # or rate 44100!or rate[:rate]*
                period_time 0 # or period_time 0!or period_time[:time]*
                period_size 0 # or period_size 0!or period_size[:size]*
                buffer_size 0 # or buffer_size 0!or buffer_size[:size]*
        }
}
```

Function reference

- snd_pcm_linear_open()
- snd_pcm_linear_open()

Plugin: Mu-Law

This plugin converts Mu-Law samples to linear or linear to Mu-Law samples from master Mu-Law conversion PCM to given slave PCM. The channel count, format and rate must match for both of them.

```
pcm.name {
        type mulaw            # Mu-Law conversion PCM
        slave PCM             # Slave PCM
        slave {               # PCM definition
                pcm "hw:0,0"  # or pcm "plughw:0,0"
                rate 44100    # or rate 44100!or rate[:rate]*
                period_time 0 # or period_time 0!or period_time[:time]*
                period_size 0 # or period_size 0!or period_size[:size]*
                buffer_size 0 # or buffer_size 0!or buffer_size[:size]*
        }
}
```

Function reference

- snd_pcm_mulaw_open()
- snd_pcm_mulaw_open()

Plugin: A-Law

This plugin converts A-Law samples to linear or linear to A-Law samples from master A-Law conversion PCM to given slave PCM. The channel count, format and rate must match for both of them.

```
pcm.name {
        type alaw             # A-Law conversion PCM
        slave PCM             # Slave PCM
        slave {               # PCM definition
                pcm "hw:0,0"  # or pcm "plughw:0,0"
                rate 44100    # or rate 44100!or rate[:rate]*
                period_time 0 # or period_time 0!or period_time[:time]*
                period_size 0 # or period_size 0!or period_size[:size]*
                buffer_size 0 # or buffer_size 0!or buffer_size[:size]*
        }
}
```

Function reference

- snd_pcm_alaw_open()
- snd_pcm_alaw_open()

Plugin: Route & Volume

This plugin converts channels and applies volume during the conversion. The channel count and rate must match for both of them.

SCHANNEL can be a channel name instead of a number (e.g.'L', LFE'). If ttp, a matching channel map will be selected for the slave.

```
pcm.name {
        type route            # Route conversion PCM
        slave PCM             # Slave name
        slave {               # PCM definition
                pcm "hw:0,0"  # or pcm "plughw:0,0"
                rate 44100    # or rate 44100!or rate[:rate]*
                period_time 0 # or period_time 0!or period_time[:time]*
                period_size 0 # or period_size 0!or period_size[:size]*
                buffer_size 0 # or buffer_size 0!or buffer_size[:size]*
        }
        [ttable [CSCHANNEL CVALUE]* # Transfer table(s) for dsbmixmat (sequence of schannels/ttables)
        [scount INTEGER]      # Number of channels
        [tstrict yes[:1] no[:0]] # Strict route (0.9 - 1.0)
}
```

Function reference

- snd_pcm_route_open()
- snd_pcm_route_open()

Plugin: Rate

This plugin converts stream rate. The input and output formats must be linear.

```
pcm.name {
        type rate             # Rate PCM
        slave PCM             # Slave PCM
        slave {               # PCM definition
                pcm "hw:0,0"  # or pcm "plughw:0,0"
                rate 44100    # or rate 44100!or rate[:rate]*
                period_time 0 # or period_time 0!or period_time[:time]*
                period_size 0 # or period_size 0!or period_size[:size]*
                buffer_size 0 # or buffer_size 0!or buffer_size[:size]*
        }
        [ttable [CSCHANNEL CVALUE]* # Transfer table(s) for dsbmixmat (sequence of schannels/ttables)
        [scount INTEGER]      # Number of channels
        [tstrict yes[:1] no[:0]] # Strict route (0.9 - 1.0)
}
```

Function reference

- snd_pcm_rate_open()
- snd_pcm_rate_open()

Automatic conversion plugin

This plugin converts channels, rate and format on request.

```
pcm.name {
        type plug             # Automatic conversion PCM
        slave PCM             # Slave name
        slave {               # PCM definition
                pcm "hw:0,0"  # or pcm "plughw:0,0"
                rate 44100    # or rate 44100!or rate[:rate]*
                period_time 0 # or period_time 0!or period_time[:time]*
                period_size 0 # or period_size 0!or period_size[:size]*
                buffer_size 0 # or buffer_size 0!or buffer_size[:size]*
        }
        [route_policy STR     # Type of route converter
        [route_converter [STR STR]] # Route converter
        [tstrict yes[:1] no[:0]] # Strict route (0.9 Tweak from defaults pcm.rate_converter
}
```

Function reference

- snd_pcm_plug_open()
- snd_pcm_plug_open()

Plugin: File

This plugin dumps contents of a PCM stream to file or pipe the stream to a command, and optionally uses an existing file as an input data source (i.e., "virtual mic").

```
pcm.name {
        type file             # File PCM
        slave PCM             # Slave PCM
        slave {               # PCM definition
                pcm "hw:0,0"  # or pcm "plughw:0,0"
                rate 44100    # or rate 44100!or rate[:rate]*
                period_time 0 # or period_time 0!or period_time[:time]*
                period_size 0 # or period_size 0!or period_size[:size]*
                buffer_size 0 # or buffer_size 0!or buffer_size[:size]*
        }
        file STR              # Output file name
        [ finfo yes[:1] no[:0] # Dump information
        [ fsync yes[:1] no[:0] # Call fsync after each transfer
        [ infile STR          # Input file name
        [ ipipe yes[:1] no[:0] # Input file is a pipe
        [ ipipe_delay INTEGER] # Input file read delay (usec, def. 0000)
}
```

Function reference

- snd_pcm_file_open()
- snd_pcm_file_open()

Plugin: Multiple streams to One

This plugin converts multiple streams to one.

```
pcm.name {
        type multi            # Multiple streams conversion PCM
        slave PCM             # Slave PCM name
        slave {               # PCM definition
                pcm STR       # Slave name
                [channels INTEGER # Slave PCM definition
                [rate INTEGER]
                [ bindings.N INTEGER # Bindings table
                [channels INTEGER]
                [rate INTEGER]
        }
        [ bindings.N INTEGER  # Bindings table
        [channels INTEGER]    # Slave PCM definition
        [rate INTEGER]
}
```

Function reference

- snd_pcm_multi_open()
- snd_pcm_file_open()

Plugin: Share

This plugin allows sharing of multiple channels with mmap clients. The access to each channel is exclusive (samples are not mixed together). It means, if the channel one is used with first client, the second client cannot access the channel one until the first client closes the channel one.

The difference from dsnoop plugin is that share plugin requires the server program "sharem", while dsnoop plugin doesn't need the explicit server but access to the shared buffer.

```
pcm.name {
        type share            # Share PCM
        slave PCM             # Slave PCM
        slave {               # PCM definition
                pcm "hw:0,0"  # or pcm "plughw:0,0"
                rate 44100    # or rate 44100!or rate[:rate]*
                period_time 0 # or period_time 0!or period_time[:time]*
                period_size 0 # or period_size 0!or period_size[:size]*
                buffer_size 0 # or buffer_size 0!or buffer_size[:size]*
        }
        bindings [ STR INTEGER ] # Share channel INT for client channel S
}
```

Function reference

- snd_pcm_share_open()
- snd_pcm_share_open()

Plugin: hooks

This plugin is used to call some hook function when the plugin is opened, modified or closed. Typically, it is used to change control values for a certain sets specially for the PCM (see the example below).

```
pcm.name {
        type hooks            # Arbitrary arguments
}

ctl.name {
        type hooks            # Arbitrary arguments
        [ hook.N STR ]        # Hook function name (optional)
        [ hook.N STR STR ]    # Hook function name (with hook_args)
        [ hook_extra STR ]    # Arbitrary function
}

# a simple example
pcm.hw {
        type hw
        card 0
        [ hook.0 STR ]        # PCM hook hooks
        [ hook.1 STR ]        # Arbitrary function
        [ hook.2 STR ]
}

# a complex example
pcm.name {
        type hw               # PCM hooks hooks
        card 0                # Card name or number
        [ hook.0 STR ]        # Arbitrary function
        [ hook.1 STR ]        # Arbitrary function
        [ hook.2 STR ]
        [ hook.3 STR ]        # Arbitrary function
        [ hook.4 STR ]        # Arbitrary function
        [ hook.5 STR ]        # Arbitrary function
        [ hook.6 STR ]        # Arbitrary function
        [ hook.7 STR ]        # Arbitrary function
        [ hook.8 STR ]        # Arbitrary function
        [ hook.9 STR ]        # Arbitrary function
        [ hook name {we pcm hooks}
        [ hook definition {we pcm hooks}
        [ hook_extra {we pcm hooks}
}
```

Examples:

```
hooks {
        type ctl_elems
        hook.0 {
                type sysfs
                sysfs /sys/kernel/fipsdev/enabled
                [ hook_extra STR ]
                [ hook_extra {}
                [ hook_extra {}
                [ hook_extra {}
                [ hook_extra {}
}

# Hook "/bin/echo 0 > /proc/asound/card0/pcm0p/sub0/hw_free"
# Hook "/bin/echo 1 > /proc/asound/card0/pcm0p/sub0/hw_free"
hooks {
        type ctl_elems
        hook.0 {
                type sysfs
                sysfs /proc/asound/card0/pcm0p/sub0/hw_free
                [ hook_extra {}
                [ hook_extra {}
}

# Hook "/bin/echo 0 > /proc/asound/card0/pcm0p/sub0/hw_free"
# Hook "/bin/echo 1 > /proc/asound/card0/pcm0p/sub0/hw_free"
# Hook "/bin/echo 2 > /proc/asound/card0/pcm0p/sub0/hw_free"
hooks {
        type ctl_elems
        hook.0 {
                type sysfs
                sysfs /proc/asound/card0/pcm0p/sub0/hw_free
                [ hook_extra {}
                [ hook_extra {}
                [ hook_extra {}
}
```

Function reference

- snd_pcm_hw_hook_open()
- snd_pcm_hw_hook_open()

Plugin: Linear conversion

This plugin provides direct mixing of multiple streams. The resolution for 32-bit mixing is only 24-bit. The low significant bytes is filled with zeros. The extra 8 bits are used for the saturation.

```
pcm.name {
        type dmix             # Direct mix
        [ ipc_key INTEGER ]   # Unique IPC key
        [ ipc_key_add_uid yes[:1] no[:0] ] # Add current UID to unique IPC key
        [ ipc_perm INTEGER ]  # IPC permissions (default 0600)
        [ ipc_gidlock yes[:1] no[:0] ] # Lock IPC resources for group
        [ ipc_uidlock yes[:1] no[:0] ] # Lock IPC resources for user
        [ slave STR ]         # Slave name
        [ slave {             # PCM definition
                [ pcm STR ]   # Slave PCM
                [ rate INTEGER ] # Slave rate
                [ format STR ] # Slave format
                [ channels INTEGER ] # Slave channels
                [ period_time INTEGER ] # Slave period time
                [ period_size INTEGER ] # Slave period size
                [ buffer_time INTEGER ] # Slave buffer time
                [ buffer_size INTEGER ] # Slave buffer size
                [ periods INTEGER ] # Slave periods
                [ buffer_to INTEGER ] # Slave buffer timeout
                [ slowpoke_to INTEGER ] # Slave slowpoke timeout
                [ period_event yes[:1] no[:0] ] # Slave period event
                [ disconnect yes[:1] no[:0] ] # Slave disconnection
                [ hwshare yes[:1] no[:0] ] # Slave hardware share
                [ minimum INTEGER ] # Slave minimum
                [ resample yes[:1] no[:0] ] # Slave resampling
                [ channels_count INTEGER ] # Slave channels count
                [ rate_precision INTEGER ] # Slave rate precision
                [ rate_in_enum yes[:1] no[:0] ] # Slave rate in enumeration
                [ rate_table [INTEGER]* ] # Slave rate table
                [ rate_table_1 [INTEGER]* ] # Slave rate table 1
                [ rate_table_2 [INTEGER]* ] # Slave rate table 2
                [ rate_table_3 [INTEGER]* ] # Slave rate table 3
                [ rate_table_4 [INTEGER]* ] # Slave rate table 4
                [
```