libsndp Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Overview

Specifications

libsndp is a library that enables sound data created with the PSP™ (PlayStation®Portable) sound artist tools to be used by a title application. The data and most of the functions of libsndp for PSP™ and libsndp for PlayStation®Vita are compatible.

libsndp also provides a voice management system and SMF (Standard MIDI File) sequencer functions.

libsndp is built on top of libsas and operates the SAS with one unit of granularity.

Although one unit of granularity is normally 256 samples, (when initialized with `sceSsInit()`), one unit of granularity can also be specified (when initialized with `sceSsInitWithGrain()`).

Files

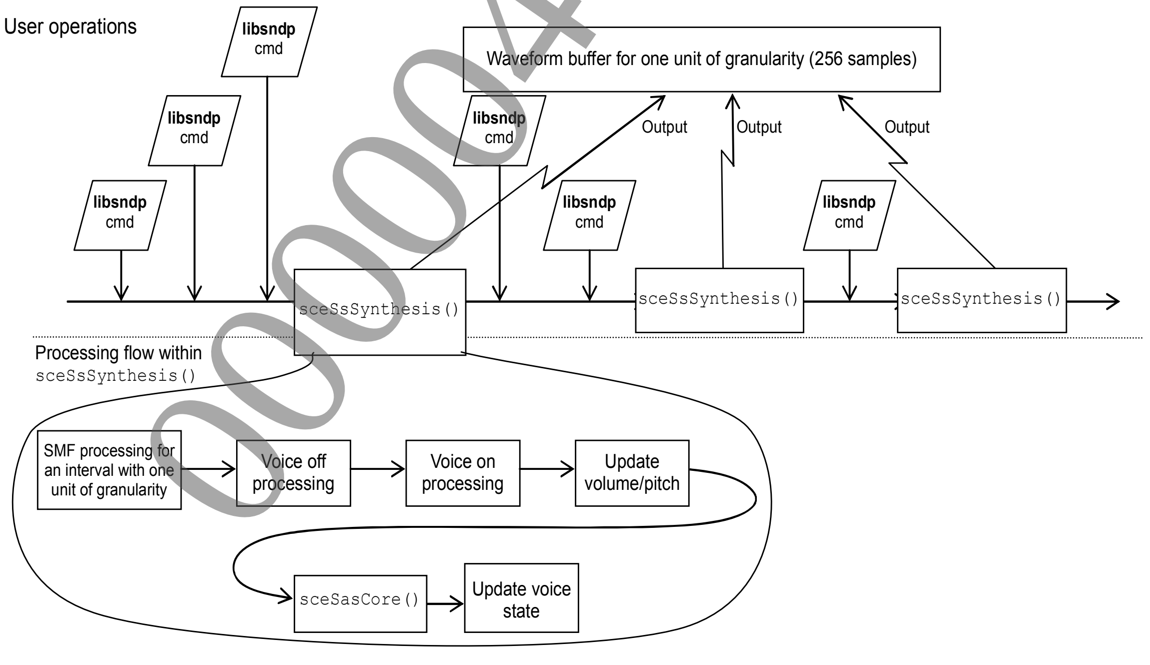
The following files are used by libsndp.

Filename	Description
libsndp.h	Library header file
sdphd.h	PHD data header file
libSceSndp.a	Library file

Processing Flow Summary

The following figure shows the overall libsndp processing flow.

Figure 1 Overall libsndp Processing Flow



libsndp invokes `sceSsSynthesis()` to perform batch processing for commands it receives. `sceSsSynthesis()` performs the desired operations and outputs the final waveform to an output buffer.

Since the unit of processing is the number of one unit of granularity samples, the contexts of the commands that libsndp receives are all lined up at the beginning of the output waveform.

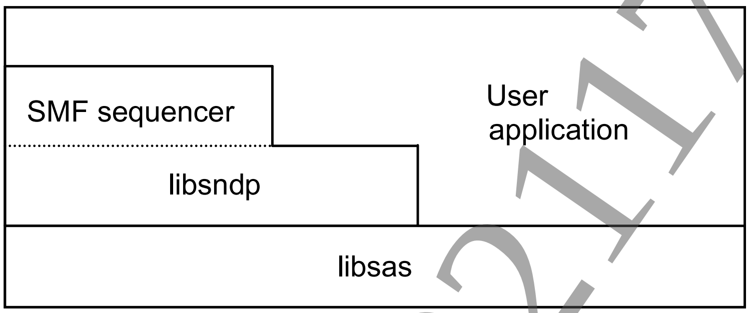
Processing performed by the SMF sequencer is absorbed by `sceSsSynthesis()` within an interval corresponding to a single unit of granularity. The results generated during that interval are also all lined up at the beginning of the output waveform.

Library Configuration

libsndp has the configuration shown below.

libsndp has functions for getting parameters from a PHD and functions for using libsas APIs to perform various operations.

Figure 2 libsndp Configuration



Reference Materials

For details about the sound data formats (PHD/PBD) used by libsndp, refer to the document entitled, "PHD/PBD Formats."

For details about SMF, refer to a commercially available standard reference manual.

2 Functions

Voice Management System

To make the best use of a limited number of voices, libsndp has a system for automatically assigning the most suitable voice. Specifically, when `sceSsNoteOn()`, `sceSsKeyOnByTone()`, and `sceSsNoteOnByTone()` are used to produce sound, voice management system is used to assign the voice.

Although voices are basically assigned when they are not producing sound, if all voices are being used, the basic operation is to assign a voice from among the voices which are in the most suitable state for being assigned, based on the priorities that were set in the PHD.

Voice state is one criteria used to determine assignment. Voice states are ordered by assigned priority from highest to lowest.

The voice state can be referenced by the `sceSsVoiceGetStatus()` libsndp function.

For details about each voice state, refer to the "libsndp Reference" document.

State	Priority
`SCE_SNDP_VOICESTATUS_IDLE`	High
`SCE_SNDP_VOICESTATUS_RELEASE`	↑
`SCE_SNDP_VOICESTATUS_SUSTAIN`	↓
`SCE_SNDP_VOICESTATUS_BUSY`	Low

Priorities are handled according to the following algorithm. If the priorities of all voices that are in a given candidate state are higher than the priority of the new sound to be generated, the next sequential state is examined. If the priorities of all voices end up being higher than the priority of the new sound to be generated, then the new sound will not be generated.

In addition, if a value other than 0 is set to the group set in the PHD, the sound generation of the same group number for which sound is being generated will be stopped. Tones with a set group number value other than 0 only generate one sound.

The voice state changes when commands are executed because of commands that start and stop voice generation. Note that if `sceSsSynthesis()` causes voice generation to be stopped, any affected voices will transition to IDLE state.

If sound generation is started, sustain = on is set, and sound generation is stopped for a given voice, the voice will transition to sustain state during the stop command. If the voice is already in sustain state, when sound generation is stopped, the voice will transition to RELEASE state.

However, when the operations performed for a voice consist of generating sound and stopping sound production (without setting sustain state), when `sceSsSynthesis()` is executed, no sound will be produced.

The voice state in effect immediately before `sceSsSynthesis()` is executed, is valid for all commands issued for that voice.

The voice management system maintains a state that is most appropriate for operating the SAS. This includes maintaining the state of commands issued for a voice and maintaining the conditions in which voices are assigned.

However, this is not necessarily the case when sound is produced by directly specifying a voice.

Note that when a voice is removed from libsndp management by `sceSsVoiceSetReserveMode()`, the voice state will change to `SCE_SNDP_VOICESTATUS_RESERVED` and the voice will no longer be available to be assigned. A voice paused with `sceSsVoiceSetPause()` will also not be available to be assigned.

Sound Generation and Operations

To generate sound, libsndp uses a key-on ID that must be set to identify the sound source.

This is for specifying the original sound generation when you want to perform separate operations for individual sound generation even when producing sound using the same content, with regard to producing sound using the same tone parameters or using note on.

For example, if three objects A, B, and C produce sound using the same tone parameters and you want to change the orientation of only sound A, you can assign different key-on IDs to A, B, and C, identify the sound producing voices from the key-on IDs, and perform separate operations for those voices.

Note that key-on IDs from 0x80000000 to 0xFFFFFFFF are reserved by libsndp.

SMF (Standard MIDI File) Sequencer

libsndp provides internal SMF (Standard MIDI File) sequencer functions.

The SMF sequencer can conduct multiple performances simultaneously. However, note that if a large number of MIDI events are included during the same unit of granularity, processing will be delayed. For that reason, deleting events that are ignored by the SMF sequencer and shifting a large number of events by a suitable interval within the same unit of granularity are recommended.

Since libsndp uses an interval of one unit of granularity of SAS, for 60 BPM, the resolution is approximately 172. If an SMF contains extremely short notes that are played at a fast tempo, the sound will be quantized.

Note that because the final content of the granularity for operations with control change or pitch bend is valid, sudden changes due to the final value of the value of the pitch operation and volume operation, in particular, can result in noise generation.

The SMF sequencer recognizes only supported MIDI events. Unsupported MIDI events are ignored.

The following events are supported.

- Note off (8nH)
- Note on (9nH)
- Control change (BnH)
  - Channel volume (formerly main volume) (07H)
  - Panpot (0AH)
  - Expression controller (0BH)
  - Hold (damper pedal, sustain) (40H)
  - Data entry (MSB) (06H)
  - NRPN (MSB) (63H)
- Program change (CnH)
- Pitch bend change (EnH)

Note that the following functions can be used by NRPN.

NRPN	Function
B0H, 63H, 00H	Set loop start
B0H, 06H, nnH	Set loop count (if nn = 00h, loop continuously)
B0H, 63H, 01H	Set loop end

The loop count must be set between the loop start setting and loop end setting. Although multiple loops can be set, nested structures are not supported.

If there are multiple loop start settings before the loop end setting, only the last loop start setting will be valid.

Also, if loop end is set without setting a loop count, no looping will be performed.

To clearly indicate the initial sound to be produced, you must make sure that a program change is included before the first note on event.

Compatibility with PSP™

The only differences with the API for PSP™ are the initialization/termination functions associated with SAS API changes and the Bind functions.

Regarding the data format, PHD, PBD, PEF, and SMF files for PSP™ can be used as is.

The reference pitch sampling frequency can be specified for the `sceSsBindSoundData()` function.

When PBD data (created in a 44.1-KHz environment) for PSP™ is played back at 48 KHz, the pitch amplitude can vary compared to a PSP™ environment and thereby affect the sound quality.

In addition, the playback tempo of SMF is also based on the playback timing, so the playback sampling frequency is needed to be specified with `sceSsSMFBind()`.