SunVox library for developers

2022.03.05

Download

The latest release: sunvox_lib-2.0e.zip

Overview

SunVox is a powerful modular synthesizer and pattern-based sequencer (tracker).
SunVox Library is the main part of the SunVox engine without a graphical interface.

Using this library, you can do the following:

- load and play several SunVox/XM/MOD music files simultaneously;
- play interactive/generative/microtonal music;
- play synths, apply effects;
- load samples (WAV,AIFF,XI), synths and effects created by other users;
- change any project parameters (synth controllers, pattern notes, etc.).

You can freely use it in your own products (even commercial ones). But don't forget to read the license file :)

Supported systems:

- Windows (dynamic library + C interface);
- macOS (dynamic library + C interface);
- Linux (dynamic library + C interface);
- Android (dynamic library + C interface + Java wrapper);
- iOS (static library + C interface).

It's also available for Web (JS), and it's a built-in library in Pixilang programming language since v3.8.

Hello SunVox Lib

Here are examples of minimal code for different platforms.
Other examples (more complex projects) can be found in the archive with the library.

C + dynamic lib (Windows, Linux, macOS):

```
#define SUNVOX_MAIN /* We are using a dynamic lib. SUNVOX_MAIN adds implementation of sv_load_dll()/sv_unload_dll() */
#include "sunvox.h"
int main()
{
    if( sv_load_dll() ) return 1;
    int ver = sv_init( 0, 44100, 2, 0 );
    if( ver >= 0 )
    {
        sv_open_slot( 0 );
        /*
        The SunVox is initialized.
        Slot 0 is open and ready for use.
        Then you can load and play some files in this slot.
        */
        sv_close_slot( 0 );
        sv_deinit();
    }
    sv_unload_dll();
    return 0;
}
```

C + static lib (iOS):

```
#define SUNVOX_STATIC_LIB /* We are using a static lib. SUNVOX_STATIC_LIB tells the compiler that all functions should be inclu
#include "sunvox.h"
//App init:
int ver = sv_init( 0, 44100, 2, 0 );
if( g_sunvox_lib_ver >= 0 )
{
    sv_open_slot( 0 );
    /*
    The SunVox is initialized.
    Slot 0 is open and ready for use.
    Then you can load and play some files in this slot.
    */
}
//App deinit:
sv_close_slot( 0 );
sv_deinit();
```

Java + dynamic lib (Android):

```
import nightradio.sunvoxlib.SunVoxLib;
//App init:
int ver = SunVoxLib.init( null, 44100, 2, 0 );
if( ver >= 0 )
{
    SunVoxLib.open_slot( 0 );
    // The SunVox is initialized.
    // Slot 0 is open and ready for use.
    // Then you can load and play some files in this slot.
}
//App deinit:
SunVoxLib.close_slot( 0 );
SunVoxLib.deinit();
```

JS + static lib:

```
<html>
<head></head>
<script src="lib/sunvox.js"></script>
<script src="lib/sunvox_lib_loader.js"></script>
<script>
svlib.then( function(Module) {
    var ver = sv_init( 0, 44100, 2, 0 );
    if( ver >= 0 )
    {
        sv_open_slot( 0 );
        // The SunVox is initialized.
        // Slot 0 is open and ready for use.
        // Then you can load and play some files in this slot.
    }
} );
</script>
<body></body>
</html>
```

Pixilang:

```
sv = sv_new()
if sv >= 0
{
    // The SunVox object (sv) is created.
    // Then you can use this object (similar to the slot on other platforms) to load and play SunVox files.
    sv_remove( sv )
}
```

Python + dynamic lib (Linux):

```
import ctypes
import time

# get script directory
import os
scriptpath= os.path.realpath(__file__)
scriptdir = os.path.dirname(scriptpath)
# construct full path to sunvox lib
libname=os.path.join(scriptdir,"sunvox.so")

if __name__ == "__main__":
    # Load the shared library into ctypes
    svlib = ctypes.CDLL(libname)

    # CONNECT TO SOUND SYSTEM
    svlib.sv_init.restype=ctypes.c_int32
    ver = svlib.sv_init(None, 44100, 2, 0 )
    print (f"Init Sound succeeded!") if ver>=0 else print (f"Link Sound failed, error:{ver}")

    if( ver >= 0 ):
        # REQUEST SLOT
        slotnr=0
        success=svlib.sv_open_slot(slotnr)
        print (f"Open slot succeeded!") if success==0 else print (f"Open slot failed, error:{success}")

        # LOAD FILE
        svfile=os.path.join(scriptdir,"test.sunvox")
        bsvfile = svfile.encode('utf-8')
        success = svlib.sv_load(slotnr, ctypes.c_char_p(bsvfile))
        print (f"Open file succeeded!") if success==0 else print (f"Open file failed, error:{success}")

        # SET VOLUME
        svlib.sv_volume(slotnr,256)

        # START PLAY
        success = svlib.sv_play_from_beginning(slotnr)
        print (f"Play file succeeded!") if success==0 else print (f"Play file failed, error:{success}")

        # LET PLAY FOR 5 SECONDS
        time.sleep(5)

        # STOP PLAY
        svlib.sv_stop(slotnr)

        # CLOSE SLOT
        svlib.sv_close_slot(slotnr)

        # RELEASE SOUND SYSTEM
        svlib.sv_deinit()
```

Constants

Note commands

- 1..127 - note number;
- NOTECMD_NOTE_OFF;
- NOTECMD_ALL_NOTES_OFF - send "note off" to all modules;
- NOTECMD_CLEAN_SYNTHS - put all modules into standby state (stop and clear all internal buffers);
- NOTECMD_STOP;
- NOTECMD_PLAY;
- NOTECMD_SET_PITCH - set the pitch specified in column XXYY, where 0x0000 - highest possible pitch, 0x7800 - lowest pitch (note C0); one semitone = 0x100.

Pitch conversion formulas for NOTECMD_SET_PITCH:

- from SunVox pitch to Hz: freq = pow(2, (30720 - pitch) / 3072) * 16.333984375
- from Hz to SunVox pitch: pitch = 30720 - log2(freq / 16.333984375) * 3072

Flags for sv_init()

- SV_INIT_FLAG_NO_DEBUG_OUTPUT;
- SV_INIT_FLAG_USER_AUDIO_CALLBACK - offline mode: system-dependent audio stream will not be created; user must call sv_audio_callback() to get the next piece of sound stream;
- SV_INIT_FLAG_OFFLINE - same as SV_INIT_FLAG_USER_AUDIO_CALLBACK;
- SV_INIT_FLAG_AUDIO_INT16 - desired sample type of the output sound stream : int16_t;
- SV_INIT_FLAG_AUDIO_FLOAT32 - desired sample type of the output sound stream : float; the actual sample type may be different, if SV_INIT_FLAG_USER_AUDIO_CALLBACK is not set;
- SV_INIT_FLAG_ONE_THREAD - audio callback and song modification are in single thread; use it with SV_INIT_FLAG_USER_AUDIO_CALLBACK only.

In Pixilang only the following flags are available: SV_INIT_FLAG_OFFLINE, SV_INIT_FLAG_ONE_THREAD.

Flags for sv_get_time_map()

- SV_TIME_MAP_SPEED;
- SV_TIME_MAP_FRAMECNT;

Flags for sv_get_module_flags()

- SV_MODULE_FLAG_EXISTS;
- SV_MODULE_FLAG_EFFECT;
- SV_MODULE_FLAG_MUTE;
- SV_MODULE_FLAG_SOLO;
- SV_MODULE_FLAG_BYPASS;
- SV_MODULE_INPUTS_OFF;
- SV_MODULE_INPUTS_MASK;
- SV_MODULE_OUTPUTS_OFF;
- SV_MODULE_OUTPUTS_MASK;

Number of the module inputs = (flags & SV_MODULE_INPUTS_MASK) >> SV_MODULE_INPUTS_OFF
Number of the module outputs = (flags & SV_MODULE_OUTPUTS_MASK) >> SV_MODULE_OUTPUTS_OFF

Other

NULL - null pointer or null reference. Called differently in different languages:

- C: NULL;
- Java, JS: null;
- Pixilang: -1.

Functions

Main

sv_load_dll()
sv_unload_dll()

(C only)
Load/unload the library (for dynamic (shared) version only: DLL, SO, DYLIB, etc.).

Return value: 0 (success) or negative error code.

Example:

```
#define SUNVOX_MAIN /* We are using a dynamic lib. SUNVOX_MAIN adds implementation of sv_load_dll()/sv_unload_dll() */
#include "sunvox.h"
int main()
{
    if( sv_load_dll() ) return 1;
    int ver = sv_init( 0, 44100, 2, 0 );
    if( ver >= 0 )
    {
        sv_open_slot( 0 );
        sv_close_slot( 0 );
        sv_deinit();
    }
    sv_unload_dll();
    return 0;
}
```

sv_init()
sv_deinit()

Global sound system initialization and deinitialization.

Prototypes:

- C:
  - int sv_init(const char* config, int sample_rate, int channels, uint32_t flags);
  - int sv_deinit(void);
- Java:
  - int init(String config, int sample_rate, int channels, int flags);
  - int deinit();
- JS:
  - sv_init(config, sample_rate, channels, flags);
  - sv_deinit();
- Pixilang: see sv_new() and sv_remove().

Parameters:

- config - string with additional configuration in the following format: "option_name=value|option_name=value"; or NULL for auto config; example: "buffer=1024|audiodriver=alsa|audiodevice=hw:0,0";
- sample_rate - desired sample rate (Hz); min - 44100; the actual rate may be different, if SV_INIT_FLAG_OFFLINE is not set;
- channels - only 2 supported now;
- flags - set of flags SV_INIT_FLAG_*;

sv_init() return value: negative error code or SunVox engine version ((major<<16) + (minor<<8) + minor2); example: 0x010906 for v1.9.6.
sv_deinit() return value: 0 (success) or negative error code.

sv_new()
sv_remove()

(Pixilang only)
Create a new SunVox engine object or remove an existing object.

Prototypes:

- Pixilang:
  - sv_new(sample_rate, flags);
  - sv_remove(sv);

Parameters:

- sample_rate - desired sample rate (Hz); min - 44100; the actual rate may be different, if SV_INIT_FLAG_OFFLINE is not set; optional;
- flags - set of flags SV_INIT_FLAG_*; optional;
- sv - SunVox object ID.

sv_new() return value: the new SunVox object ID or negative error code.
sv_remove() return value: 0 (success) or negative error code.

sv_get_sample_rate()

Get current sampling rate (it may differ from the frequency specified in sv_init()/sv_new()).

Prototypes:

- C: int sv_get_sample_rate(void);
- Java: int get_sample_rate();
- JS: sv_get_sample_rate();
- Pixilang: sv_get_sample_rate(sv); //sv - SunVox object ID

Return value: sampling rate or negative error code.

sv_update_input()

Handle input ON/OFF requests to enable/disable input ports of the sound card (for example, after the Input module creation). Call it from the main thread only, where the SunVox sound stream is not locked.

Prototypes:

- C: int sv_update_input(void);
- Java: int update_input();
- JS: sv_update_input();

sv_audio_callback()
sv_audio_callback2()

With these functions you can ignore the built-in SunVox sound output mechanism and render to memory / file / some other sound system.
SV_INIT_FLAG_OFFLINE flag in sv_init() must be set.

sv_audio_callback() - get the next piece of SunVox audio from the Output module.
sv_audio_callback2() - send some data to the Input module and receive the filtered data from the Output module.

Prototypes:

- C:
  - int sv_audio_callback(void* buf, int frames, int latency, uint32_t out_time);
  - int sv_audio_callback2(void* buf, int frames, int latency, uint32_t out_time, int in_type, int in_channels, void* in_buf);
- Java:
  - int audio_callback(byte[] buf, int frames, int latency, int out_time);

    - int audio_callback2(byte[] buf, int frames, int latency, int out_time, int in_type, int in_channels, byte[] in_buf);
- JS:
    - sv_audio_callback(out_buf, frames, latency, out_time);
    - sv_audio_callback2(out_buf, frames, latency, out_time, in_type, in_channels, in_buf);
- Pixilang: see sv_render().

Parameters:

- buf - output buffer of type int16_t (if SV_INIT_FLAG_AUDIO_INT16 is set in sv_init()) or float (if SV_INIT_FLAG_AUDIO_FLOAT32 is set in sv_init()); stereo data will be interleaved in this buffer: LRLR... (LR is a single frame (Left+Right));
- frames - number of frames in destination buffer;
- latency - audio latency (in frames);
- out_time - buffer output time (in system ticks);
- in_type - input buffer type: 0 - int16_t (16bit integer); 1 - float (32bit floating point);
- in_channels - number of input channels;
- in_buf - input buffer; stereo data must be interleaved in this buffer: LRLR...

Return value: 0 - silence, the output buffer is filled with zeros; 1 - the output buffer is filled.

Example 1 (simplified, without accurate time sync) - suitable for most cases:

```
sv_audio_callback( buf, frames, 0, sv_get_ticks() );
```

Example 2 (accurate time sync) - when you need to maintain exact time intervals between incoming events (notes, commands, etc.):

```
user_out_time = ... ; //output time in user time space (depends on your own implementation)
user_cur_time = ... ; //current time in user time space
user_ticks_per_second = ... ; //ticks per second in user time space
user_latency = user_out_time - user_cur_time; //latency in user time space
uint32_t sunvox_latency = ( user_latency * sv_get_ticks_per_second() ) / user_ticks_per_second; //latency in system time space
uint32_t latency_frames = ( user_latency * sample_rate_Hz ) / user_ticks_per_second; //latency in frames
sv_audio_callback( buf, frames, latency_frames, sv_get_ticks() + sunvox_latency );
```

sv_render()

(Pixilang only)
Similar to sv_audio_callback2(): send data to the Input module and receive data from the Output module.

Prototypes:

- Pixilang: sv_render(sv, buf, frames, latency, out_time, in_buf, in_channels);

Parameters:

- sv - SunVox object ID;
- buf - output buffer (container ID); stereo data will be interleaved in this buffer: LRLR... (LR is a single frame (Left+Right));
- frames - number of frames in destination buffer; optional;
- latency - audio latency (in frames); optional;
- out_time - buffer output time (in system ticks); optional;
- in_buf - input buffer (container ID); stereo data must be interleaved in this buffer: LRLR... ; optional;
- in_channels - number of input channels; optional.

Return value: 0 - silence, the output buffer is not filled; 1 - the output buffer is filled; 2 - silence, the output buffer is filled with zeros.

sv_open_slot()
sv_close_slot()

Open/close sound slot. Each slot can contain one independent implementation of the SunVox engine. You can use several slots simultaneously (for example, one slot for music and another for effects). Max number of slots = 16.

Prototypes:

- C:
  - int sv_open_slot(int slot);
  - int sv_close_slot(int slot);
- Java:
  - int open_slot(int slot);
  - int close_slot(int slot);
- JS:
  - sv_open_slot(slot);
  - sv_close_slot(slot);
- Pixilang: see sv_new() and sv_remove().

Parameters:

- slot - slot number.

Return value: 0 (success) or negative error code.

sv_lock_slot()
sv_unlock_slot()
sv_lock()
sv_unlock()

Lock/unlock the specified SunVox slot (or object in Pixilang). Use lock/unlock when you simultaneously read and modify SunVox data from different threads (for the same slot). Some functions (marked as "USE LOCK/UNLOCK") can't work without lock/unlock at all.

Prototypes:

- C:
  - int sv_lock_slot(int slot);
  - int sv_unlock_slot(int slot);
- Java:
  - int lock_slot(int slot);
  - int unlock_slot(int slot);
- JS:
  - sv_lock_slot(slot);
  - sv_unlock_slot(slot);
- Pixilang:
  - sv_lock(sv);
  - sv_unlock(sv);

Parameters:

- slot / sv - slot number / SunVox object ID.

Return value: 0 (success) or negative error code.

Example:

```
//Thread 1:
sv_lock_slot(0);
sv_get_module_flags(0,mod1);
sv_unlock_slot(0);

//Thread 2:
sv_lock_slot(0);
sv_remove_module(0,mod2);
sv_unlock_slot(0);
```

Project file

sv_load()
sv_load_from_memory()
sv_fload()

Load SunVox project from the file or from the memory block.

Prototypes:

- C:
  - int sv_load(int slot, const char* filename);
  - int sv_load_from_memory(int slot, void* data, uint32_t data_size);
- Java:
  - int load(int slot, String filename);
  - int load_from_memory(int slot, byte[] data);
- JS:
  - sv_load_from_memory(slot, data); //load project from byte array (Uint8Array)
- Pixilang:
  - sv_load(sv, filename);
  - sv_fload(sv, f); //load project from stream f

Parameters:

- slot / sv - slot number / SunVox object ID;
- filename - file name;
- data - byte array with the project (to load from memory);
- data_size - number of bytes to read;
- f - file stream (Pixilang only).

Return value: 0 (success) or negative error code.

sv_save()
sv_fsave()

Save SunVox project.

Prototypes:

- C:
  - int sv_save(int slot, const char* filename);
- Java:
  - int save(int slot, String filename);
- Pixilang:
  - sv_save(sv, filename);
  - sv_fsave(sv, f); //save project to stream f

Parameters:

- sv - SunVox object ID;
- filename - file name;
- f - file stream.

Return value: 0 (success) or negative error code.

Project playback

sv_play()
sv_play_from_beginning()
sv_stop()

sv_play() - play from the current position.
sv_play_from_beginning() - play from the beginning (line 0).
sv_stop(): first call - stop playing; second call - reset all SunVox activity and switch the engine to standby mode.

Prototypes:

- C:
  - int sv_play(int slot);
  - int sv_play_from_beginning(int slot);
  - int sv_stop(int slot);
- Java:
  - int play(int slot);
  - int play_from_beginning(int slot);
  - int stop(int slot);
- JS:
  - sv_play(slot);
  - sv_play_from_beginning(slot);
  - sv_stop(slot);
- Pixilang:
  - sv_play(sv, line_num); //play from the specified line number
  - sv_stop(sv);

Parameters:

- slot / sv - slot number / SunVox object ID;
- line_num - start line number (optional).

Return value: 0 (success) or negative error code.

sv_pause()
sv_resume()
sv_sync_resume()

sv_pause() - pause the audio stream on the specified slot.
sv_resume() - resume the audio stream on the specified slot.
sv_sync_resume() - wait for sync (pattern effect 0x33 on any slot) and resume the audio stream on the specified slot.

Prototypes:

- C:
  - int sv_pause(int slot);
  - int sv_resume(int slot);
  - int sv_sync_resume(int slot);
- Java:

    - int pause(int slot);
    - int resume(int slot);
    - int sync_resume(int slot);
- JS:
    - sv_pause(slot);
    - sv_resume(slot);
    - sv_sync_resume(slot);
- Pixilang:
    - sv_pause(sv);
    - sv_resume(sv);
    - sv_sync_resume(sv);

Parameters:

- slot / sv - slot number / SunVox object ID.

Return value: 0 (success) or negative error code.

sv_set_autostop()
sv_get_autostop()

Set/get autostop mode. When autostop is OFF, the project plays endlessly in a loop.

Prototypes:

- C:
    - int sv_set_autostop(int slot, int autostop);
    - int sv_get_autostop(int slot);
- Java:
    - int set_autostop(int slot, int autostop);
    - int get_autostop(int slot);
- JS:
    - sv_set_autostop(slot, autostop);
    - sv_get_autostop(slot);
- Pixilang:
    - sv_set_autostop(sv, autostop);
    - sv_get_autostop(sv);

Parameters:

- slot / sv - slot number / SunVox object ID;
- autostop: 0 - disable; 1 - enable.

sv_set_autostop() return value: 0 (success) or negative error code.
sv_get_autostop() return value: 1 - autostop enabled; 0 - autostop disabled.

sv_end_of_song()

Check if the project has finished playing.

Prototypes:

- C: int sv_end_of_song(int slot);
- Java: int end_of_song(int slot);
- JS: sv_end_of_song(slot);
- Pixilang: see sv_get_status().

Parameters:

- slot - slot number.

Return value: 0 - the project is playing now; 1 - the project is stopped now.

sv_get_status()

(Pixilang only)
Get the project playing status.

Prototypes:

- Pixilang: sv_get_status(sv).

Parameters:

- sv - SunVox object ID;

Return value: 0 - the project is stopped now; 1 - the project is playing now.

sv_rewind()

Jump to the specified position (line number on the timeline).

Prototypes:

- C: int sv_rewind(int slot, int line_num);
- Java: int rewind(int slot, int line_num);
- JS: sv_rewind(slot, line_num);
- Pixilang: sv_rewind(sv, line_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- line_num - line number on the timeline.

Return value: 0 (success) or negative error code.

sv_volume()

Set the project volume.

Prototypes:

- C: int sv_volume(int slot, int vol);
- Java: int volume(int slot, int vol);
- JS: sv_volume(slot, vol);
- Pixilang: sv_volume(sv, vol);

Parameters:

- slot / sv - slot number / SunVox object ID;
- vol - volume from 0 (min) to 256 (max 100%); negative values are ignored.

Return value: previous volume or negative error code.

sv_get_current_line()
sv_get_current_line2()

sv_get_current_line() - get current (real time) line number on the timeline.sv_get_current_line2() - get current line in fixed point format 27.5.

Prototypes:

- C:
  - int sv_get_current_line(int slot);
  - int sv_get_current_line2(int slot);
- Java:
  - int get_current_line(int slot);
  - int get_current_line2(int slot);
- JS:
  - sv_get_current_line(slot);
  - sv_get_current_line2(slot);
- Pixilang:
  - sv_get_current_line(sv);
  - sv_get_current_line2(sv);

Parameters:

- slot / sv - slot number / SunVox object ID.

Return value: current line number (playback position) on the timeline.

sv_get_current_signal_level()

Get current (real time) signal level from the Output module.

Prototypes:

- C: int sv_get_current_signal_level(int slot, int channel);
- Java: int get_current_signal_level(int slot, int channel);
- JS: sv_get_current_signal_level(slot, channel);
- Pixilang: sv_get_current_signal_level(sv, channel);

Parameters:

- slot / sv - slot number / SunVox object ID;
- channel: 0 - left; 1 - right.

Return value: current signal level (from 0 to 255).

Project info

sv_get_song_name()
sv_get_name()

Get the project name.

Prototypes:

- C: const char* sv_get_song_name(int slot);
- Java: String get_song_name(int slot);
- JS: sv_get_song_name(slot);
- Pixilang: sv_get_name(sv);

Parameters:

- slot / sv - slot number / SunVox object ID.

Return value: project name or NULL.
In Pixilang the returned string container must be removed manually.

sv_get_song_bpm()
sv_get_song_tpl()
sv_get_bpm()
sv_get_tpl()

Get the project BPM (Beats Per Minute) or TPL (Ticks Per Line).

Prototypes:

- C:
  - int sv_get_song_bpm(int slot);
  - int sv_get_song_tpl(int slot);
- Java:
  - int get_song_bpm(int slot);
  - int get_song_tpl(int slot);
- JS:
  - sv_get_song_bpm(slot);
  - sv_get_song_tpl(slot);
- Pixilang:
  - sv_get_bpm(sv);
  - sv_get_tpl(sv);

Parameters:

- slot / sv - slot number / SunVox object ID.

Return value: project BPM or TPL.

sv_get_song_length_frames()
sv_get_song_length_lines()
sv_get_length_frames()
sv_get_length_lines()

Get the the project length in frames or lines.
A frame is a pair of audio signal samples (left and right channel amplitudes). The sample rate 44100 Hz means, that you hear 44100 frames per second.

Prototypes:

- C:
  - uint32_t sv_get_song_length_frames(int slot);
  - uint32_t sv_get_song_length_lines(int slot);
- Java:
  - int get_song_length_frames(int slot);
  - int get_song_length_lines(int slot);
- JS:
  - sv_get_song_length_frames(slot);
  - sv_get_song_length_lines(slot);
- Pixilang:
  - sv_get_length_frames(sv);
  - sv_get_length_lines(sv);

Parameters:

- slot / sv - slot number / SunVox object ID.

Return value: project length in frames or lines.

sv_get_time_map()

Get the project time map.

Prototypes:

- C: int sv_get_time_map(int slot, int start_line, int len, uint32_t* dest, int flags);
- Java: int get_time_map(int slot, int start_line, int len, int[] dest, int flags);
- JS: sv_get_time_map(slot, start_line, len, dest, flags); //dest is Uint32Array
- Pixilang: sv_get_time_map(sv, start_line, len, dest, flags); //dest is a container of type INT32

Parameters:

- slot / sv - slot number / SunVox object ID;
- start_line - first line to read (usually 0);
- len - number of lines to read;
- dest - pointer to the buffer (size = len*sizeof(uint32_t)) for storing the map values;
- flags:
  - SV_TIME_MAP_SPEED: dest[X] = BPM | (TPL << 16) (speed at the beginning of line X);
  - SV_TIME_MAP_FRAMECNT: dest[X] = frame counter at the beginning of line X.

Return value: 0 (success) or negative error code.

Events

sv_set_event_t()

Set the timestamp of events to be sent by sv_send_event().
Every event you send has a timestamp - this is the time when the event was generated (for example, when a key was pressed).
out_t = final time when the event can be heard from the speakers.
If timestamp is zero: out_t = as quickly as possible.
If timestamp is nonzero: out_t = timestamp + sound latancy * 2.

Prototypes:

- C: int sv_set_event_t(int slot, int set, int t);
- Java: int set_event_t(int slot, int set, int t);
- JS: sv_set_event_t(slot, set, t);
- Pixilang: sv_set_event_t(sv, set, t);

Parameters:

- slot / sv - slot number / SunVox object ID;
- set: 1 - set timestamp; 0 - reset to automatic time setting (t will be ignored; default mode);
- t - timestamp (in system ticks) for all further events.

Return value: 0 (success) or negative error code.

Examples:

```
sv_set_event_t( slot, 1, 0 ) //not specified - further events will be processed as quickly as possible
/* send some events ... */
sv_set_event_t( slot, 1, sv_get_ticks() ) //time when the events will be processed = NOW + sound latancy * 2
/* send some events ... */
```

sv_send_event()

Send an event (commands such as Note ON, Note OFF, controller change, etc.) to the SunVox engine for

further processing.

Prototypes:

- C: int sv_send_event(int slot, int track_num, int note, int vel, int module, int ctl, int ctl_val);
- Java: int send_event(int slot, int track_num, int note, int vel, int module, int ctl, int ctl_val);
- JS: sv_send_event(slot, track_num, note, vel, module, ctl, ctl_val);
- Pixilang: sv_send_event(sv, track_num, note, vel, module, ctl, ctl_val);

Parameters:

- slot / sv - slot number / SunVox object ID;
- track_num - track number (within the virtual pattern);
- note: 0 - nothing; 1..127 - note number; 128 - note off; 129, 130... - seeNOTECMD_* defines;
- vel: velocity 1..129; 0 - default;
- module: 0 (empty) or module number + 1 (1..65535);
- ctl: 0xCCEE; CC - controller number (1..255); EE - effect;
- ctl_val: value (0..32768) of the controller CC or parameter (0..65535) of the effect EE.

Return value: 0 (success) or negative error code.

Examples:

```
int m = sv_find_module( slot, "Sampler" ); //find a module named "Sampler"
sv_set_event_t( slot, 1, 0 ); //handle events as quickly as possible

//Set controller 2 (Panning) to maximum:
int ctl_num = 2;
int ctl_val = 32768;
sv_send_event( slot, 0, 0, 0, m+1, ctl_num << 8, ctl_val );

//Send Note ON to the module m:
int n = 5 * 12 + 4; //octave 5, note 4
sv_send_event( slot, 0, n+1, 129, m+1, 0, 0 ); //track 0; note n; velocity 129 (max); module m;

//Send Note OFF to the module m:
//(turn off the previous note on track 0)
sv_send_event( slot, 0, NOTECMD_NOTE_OFF, 0, 0, 0, 0 ); //track 0; module = last used on this track;

//Play the exact frequency in Hz:
//(but the actual frequency will depend the module and its parameters)

float freq = 440; //440 Hz

//method 1 (C only):
sv_send_event( slot, 0, NOTECMD_SET_PITCH, 129, m+1, 0, SV_FREQUENCY_TO_PITCH( freq ) );

//method 2:
int pitch = 30720 - log2( freq / 16.333984375 ) * 3072;
sv_send_event( slot, 0, NOTECMD_SET_PITCH, 129, m+1, 0, pitch );

//Conversion formulas:
//  from SunVox pitch to Hz: freq = pow( 2, ( 30720 - pitch ) / 3072 ) * 16.333984375;
//  from Hz to SunVox pitch: pitch = 30720 - log2( freq / 16.333984375 ) * 3072;
```

Modules

sv_new_module()

Create a new module.
USE LOCK/UNLOCK!

Prototypes:

- C: int sv_new_module(int slot, const char* type, const char* name, int x, int y, int z);
- Java: int new_module(int slot, String type, String name, int x, int y, int z);
- JS: sv_new_module(slot, type, name, x, y, z);
- Pixilang: sv_new_module(sv, type, name, x, y, z);

Parameters:

- slot / sv - slot number / SunVox object ID;
- type - string with module type; example: "Sampler";
- name - module name;
- x, y - module coordinates; center of the module view = 512,512; normal working area: 0,0 ... 1024,1024;
- z - layer number from 0 to 7.

Return value: the number of the newly created module, or negative error code.

sv_remove_module()

Remove the specified module.
USE LOCK/UNLOCK!

Prototypes:

- C: int sv_remove_module(int slot, int mod_num);
- Java: int remove_module(int slot, int mod_num);
- JS: sv_remove_module(slot, mod_num);
- Pixilang: sv_remove_module(sv, mod_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- mod_num - module number.

Return value: 0 (success) or negative error code.

sv_connect_module()
sv_disconnect_module()

Connect/disconnect two specified modules: source --> destination.
USE LOCK/UNLOCK!

Prototypes:

- C:
  - int sv_connect_module(int slot, int source, int destination);
  - int sv_disconnect_module(int slot, int source, int destination);
- Java:
  - int connect_module(int slot, int source, int destination);
  - int disconnect_module(int slot, int source, int destination);
- JS:
  - sv_connect_module(slot, source, destination);
  - sv_disconnect_module(slot, source, destination);
- Pixilang:
  - sv_connect_module(sv, source, destination);
  - sv_disconnect_module(sv, source, destination);

Parameters:

- slot / sv - slot number / SunVox object ID;

- source - source (module number);
- destination - destination (module number).

Return value: 0 (success) or negative error code.

sv_load_module()
sv_load_module_from_memory()
sv_fload_module()

Load the module or sample. Supported file formats: sunsynth, xi, wav, aiff.
For WAV and AIFF: only uncompressed PCM format is supported.

Prototypes:

- C:
  - int sv_load_module(int slot, const char* filename, int x, int y, int z);
  - int sv_load_module_from_memory(int slot, void* data, uint32_t data_size, int x, int y, int z);
- Java:
  - int load_module(int slot, String filename, int x, int y, int z);
  - int load_module_from_memory(int slot, byte[] data, int x, int y, int z);
- JS:
  - sv_load_module_from_memory(slot, data, x, y, z); //load from data (Uint8Array)
- Pixilang:
  - sv_load_module(sv, filename, x, y, z); //x,y,z - optional
  - sv_fload_module(sv, f, x, y, z); //load from file stream f

Parameters:

- slot / sv - slot number / SunVox object ID;
- filename - file name (C and Java only);
- data - byte array with file contents (C, Java and JS only);
- data_size - number of bytes in the array (C only);
- f - file stream (Pixilang only);
- x, y, z - coordinates and layer.

Return value: the number of the newly created (from file) module, or negative error code.

sv_sampler_load()
sv_sampler_load_from_memory()
sv_sampler_fload()

Load the sample (xi, wav, aiff) to the already created Sampler module.
For WAV and AIFF: only uncompressed PCM format is supported.
To replace the whole sampler - set sample_slot to -1.

Prototypes:

- C:
  - int sv_sampler_load(int slot, int sampler_module, const char* filename, int sample_slot);
  - int sv_sampler_load_from_memory(int slot, int sampler_module, void* data, uint32_t data_size, int sample_slot);
- Java:
  - int sampler_load(int slot, int sampler_module, String filename, int sample_slot);
  - int sampler_load_from_memory(int slot, int sampler_module, byte[] data, int sample_slot);
- JS:
  - sv_sampler_load_from_memory(slot, sampler_module, data, sample_slot); //load from data (Uint8Array)
- Pixilang:

    - sv_sampler_load(sv, sampler_module, filename, sample_slot); //sample_slot - optional
    - sv_sampler_fload(sv, sampler_module, f, sample_slot); //load from file stream f

Parameters:

- slot / sv - slot number / SunVox object ID;
- sampler_module - Sampler module number;
- filename - file name (C and Java only);
- data - byte array with file contents (C, Java and JS only);
- data_size - number of bytes in the array (C only);
- f - file stream (Pixilang only);
- sample_slot - slot number inside the Sampler, or -1 if you want to replace the whole module.

Return value: 0 (success) or negative error code.

sv_get_number_of_modules()

Get the number of module slots (not the actual number of modules) in the project. The slot can be empty or it can contain a module.
Here is the code to determine that the module slot X is not empty: (sv_get_module_flags(slot, X) & SV_MODULE_FLAG_EXISTS) != 0;

Prototypes:

- C: int sv_get_number_of_modules(int slot);
- Java: int get_number_of_modules(int slot);
- JS: sv_get_number_of_modules(slot);
- Pixilang: sv_get_number_of_modules(sv);

Parameters:

- slot / sv - slot number / SunVox object ID.

Return value: number of modules or negative error code.

sv_find_module()

Find a module by name.

Prototypes:

- C: int sv_find_module(int slot, const char* name);
- Java: int find_module(int slot, String name);
- JS: sv_find_module(slot, name);
- Pixilang: sv_find_module(sv, name);

Parameters:

- slot / sv - slot number / SunVox object ID;
- name - module name.

Return value: module number or -1 (module not found).

sv_get_module_flags()

Get flags of the specified module.

Prototypes:

- C: uint32_t sv_get_module_flags(int slot, int mod_num);

- Java: int get_module_flags(int slot, int mod_num);
- JS: sv_get_module_flags(slot, mod_num);
- Pixilang: sv_get_module_flags(sv, mod_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- mod_num - module number.

Return value: set of flags SV_MODULE_FLAG_* or -1 (error).

sv_get_module_inputs()
sv_get_module_outputs()

Get int[] arrays with the input/output links.
Number of input links = (module_flags & SV_MODULE_INPUTS_MASK) >> SV_MODULE_INPUTS_OFF
Number of output links = (module_flags & SV_MODULE_OUTPUTS_MASK) >> SV_MODULE_OUTPUTS_OFF
(this is not the actual number of connections: some links may be empty (value = -1))

Prototypes:

- C:
  - int* sv_get_module_inputs(int slot, int mod_num);
  - int* sv_get_module_outputs(int slot, int mod_num);
- Java:
  - int[] get_module_inputs(int slot, int mod_num);
  - int[] get_module_outputs(int slot, int mod_num);
- JS:
  - sv_get_module_inputs(slot, mod_num); //return value: Int32Array
  - sv_get_module_outputs(slot, mod_num); //return value: Int32Array
- Pixilang:
  - sv_get_module_inputs(sv, mod_num); //return value: INT32 container ID
  - sv_get_module_outputs(sv, mod_num); //return value: INT32 container ID

Parameters:

- slot / sv - slot number / SunVox object ID;
- mod_num - module number.

Return value: pointer to the int[] array with the input/output links or NULL.
In Pixilang the returned container must be removed manually.

sv_get_module_name()

Get the module name.

Prototypes:

- C: const char* sv_get_module_name(int slot, int mod_num);
- Java: String get_module_name(int slot, int mod_num);
- JS: sv_get_module_name(slot, mod_num);
- Pixilang: sv_get_module_name(sv, mod_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- mod_num - module number.

Return value: module name or NULL.
In Pixilang the returned string container must be removed manually.

sv_get_module_xy()

Get the module coordinates (XY) packed in a single uint32 value: (x & 0xFFFF) | ((y & 0xFFFF) << 16)
Normal working area: 0,0 ... 1024,1024. Center of the module view = 512,512.
In C you can use SV_GET_MODULE_XY() macro to unpack X and Y.

Prototypes:

- C: uint32_t sv_get_module_xy(int slot, int mod_num);
- Java: int get_module_xy(int slot, int mod_num);
- JS: sv_get_module_xy(slot, mod_num);
- Pixilang: sv_get_module_xy(sv, mod_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- mod_num - module number.

Return value: module coordinates (x & 0xFFFF) | ((y & 0xFFFF) << 16)

Example:

```
//Get packed XY:
int xy = sv_get_module_xy( slot, mod );
//Unpack X and Y:
int x = xy & 0xFFFF; if( x & 0x8000 ) x -= 0x10000;
int y = ( xy >> 16 ) & 0xFFFF; if( y & 0x8000 ) y -= 0x10000;
```

sv_get_module_color()

Get the module color.

Prototypes:

- C: int sv_get_module_color(int slot, int mod_num);
- Java: int get_module_color(int slot, int mod_num);
- JS: sv_get_module_color(slot, mod_num);
- Pixilang: sv_get_module_color(sv, mod_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- mod_num - module number.

Return value: module color in one of the following formats:

- Pixilang: native color value (can be used directly in graphics functions);
- other languages: 0xBBGGRR.

Example (C):

```
int rgb = sv_get_module_color( slot, mod );
int r = rgb & 0xFF;           //r = 0...255
int g = ( rgb >> 8 ) & 0xFF;  //g = 0...255
int b = ( rgb >> 16 ) & 0xFF; //b = 0...255
```

Example (Pixilang):

```
color = sv_get_module_color( sv, mod )
fbox( 0, 0, 100, 100, color )
```

sv_get_module_finetune()

Get the module relative note (transposition) and finetune (-256...0...256) packed in a single uint32 value: (relnote & 0xFFFF) | ((finetune & 0xFFFF) << 16)

Prototypes:

- C: uint32_t sv_get_module_finetune(int slot, int mod_num);
- Java: int get_module_finetune(int slot, int mod_num);
- JS: sv_get_module_finetune(slot, mod_num);
- Pixilang: sv_get_module_finetune(sv, mod_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- mod_num - module number.

Return value: module relative note (transposition) and finetune (relnote & 0xFFFF) | ((finetune & 0xFFFF) << 16)

Example:

```
//Get packed RelNote+Finetune:
int f = sv_get_module_finetune( slot, mod );
//Unpack RelNote and Finetune:
int relnote = f & 0xFFFF; if( relnote & 0x8000 ) relnote -= 0x10000;
int finetune = ( f >> 16 ) & 0xFFFF; if( finetune & 0x8000 ) finetune -= 0x10000;
```

sv_get_module_scope2()
sv_get_module_scope()

Get the currently playing piece of sound from the output of the specified module.

Prototypes:

- C: uint32_t sv_get_module_scope2(int slot, int mod_num, int channel, int16_t* dest_buf, uint32_t samples_to_read);
- Java: int get_module_scope(int slot, int mod_num, int channel, short[] dest_buf, int samples_to_read);
- JS: sv_get_module_scope2(slot, mod_num, channel, dest_buf, samples_to_read); //dest_buf must be Int16Array
- Pixilang: sv_get_module_scope(sv, mod_num, channel, dest_buf, samples_to_read); //dest_buf must be INT16 container

Parameters:

- slot / sv - slot number / SunVox object ID;
- mod_num - module number;
- channel: 0 - left; 1 - right;
- dest_buf - pointer to the buffer (int16) to store the audio fragment;
- samples_to_read - the number of samples to read from the module's output buffer.

Return value: received number of samples (may be less or equal to samples_to_read).

Example (C):

```
int16_t buf[ 1024 ];
int received = sv_get_module_scope2( slot, mod_num, 0, buf, 1024 );
//buf[ 0 ] = value of the first sample (-32768...32767);
//buf[ 1 ] = value of the second sample;
//...
//buf[ received - 1 ] = value of the last received sample;
```

sv_module_curve()

Access to the curve values of the specified module.

Prototypes:

- C: int sv_module_curve(int slot, int mod_num, int curve_num, float* data, int len, int w);
- Java: int module_curve(int slot, int mod_num, int curve_num, float[] data, int len, int w);
- JS: sv_module_curve(slot, mod_num, curve_num, data, len, w); //data must be Float32Array
- Pixilang: sv_module_curve(sv, mod_num, curve_num, data, len, w); //data must be FLOAT32 container

Parameters:

- slot / sv - slot number / SunVox object ID;
- mod_num - module number;
- curve_num - curve number;
- data - pointer to the destination or source buffer (array);
- len - number of items to read/write;
- w - mode: read (0) or write (1).

Return value: number of items processed successfully or negative error code.

Available curves (Y=CURVE[X]):

- MultiSynth:
  - 0 - X = note (0..127); Y = velocity (0..1); 128 items;
  - 1 - X = velocity (0..256); Y = velocity (0..1); 257 items;
- WaveShaper:
  - 0 - X = input (0..255); Y = output (0..1); 256 items;
- MultiCtl:
  - 0 - X = input (0..256); Y = output (0..1); 257 items;
- Analog Generator, Generator:
  - 0 - X = drawn waveform sample number (0..31); Y = volume (-1..1); 32 items;

sv_get_number_of_module_ctls()

Get the number of the module controllers.

Prototypes:

- C: int sv_get_number_of_module_ctls(int slot, int mod_num);
- Java: int get_number_of_module_ctls(int slot, int mod_num);
- JS: sv_get_number_of_module_ctls(slot, mod_num);
- Pixilang: sv_get_number_of_module_ctls(sv, mod_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- mod_num - module number.

Return value: number of the module controllers or negative error code.

sv_get_module_ctl_name()

Get the name of the specified module controller.

Prototypes:

- C: const char* sv_get_module_ctl_name(int slot, int mod_num, int ctl_num);
- Java: String get_module_ctl_name(int slot, int mod_num, int ctl_num);
- JS: sv_get_module_ctl_name(slot, mod_num, ctl_num);
- Pixilang: sv_get_module_ctl_name(sv, mod_num, ctl_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- mod_num - module number;
- ctl_num - controller number (from 0).

Return value: controller name or NULL.
In Pixilang the returned string container must be removed manually.

sv_get_module_ctl_value()

Get the value of the specified module controller.

Prototypes:

- C: int sv_get_module_ctl_value(int slot, int mod_num, int ctl_num, int scaled);
- Java: int get_module_ctl_value(int slot, int mod_num, int ctl_num, int scaled);
- JS: sv_get_module_ctl_value(slot, mod_num, ctl_num, scaled);
- Pixilang: sv_get_module_ctl_value(sv, mod_num, ctl_num, scaled);

Parameters:

- slot / sv - slot number / SunVox object ID;
- mod_num - module number;
- ctl_num - controller number (from 0);
- scaled: 0 - real value (0,1,2 etc.); 1 - scaled for the pattern column XXYY (0x0000...0x8000).

Return value: value of the specified module controller.

Patterns

sv_get_number_of_patterns()

Get the number of pattern slots (not the actual number of patterns) in the project. The slot can be empty or it can contain a pattern.
Here is the code to determine that the pattern slot X is not empty: sv_get_pattern_lines(slot, X) > 0;

Prototypes:

- C: int sv_get_number_of_patterns(int slot);
- Java: int get_number_of_patterns(int slot);
- JS: sv_get_number_of_patterns(slot);
- Pixilang: sv_get_number_of_patterns(sv);

Parameters:

- slot / sv - slot number / SunVox object ID.

Return value: number of patterns or negative error code.

sv_find_pattern()

Find a pattern by name.

Prototypes:

- C: int sv_find_pattern(int slot, const char* name);
- Java: int find_pattern(int slot, String name);
- JS: sv_find_pattern(slot, name);
- Pixilang: sv_find_pattern(sv, name);

Parameters:

- slot / sv - slot number / SunVox object ID;
- name - pattern name.

Return value: pattern number or -1 (pattern not found).

sv_get_pattern_x()

Get the X (line number) coordinate of the pattern on the timeline.

Prototypes:

- C: int sv_get_pattern_x(int slot, int pat_num);
- Java: int get_pattern_x(int slot, int pat_num);
- JS: sv_get_pattern_x(slot, pat_num);
- Pixilang: sv_get_pattern_x(sv, pat_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- pat_num - pattern number.

Return value: X (line number) coordinate of the pattern on the timeline.

sv_get_pattern_y()

Get the Y coordinate of the pattern on the timeline.

Prototypes:

- C: int sv_get_pattern_y(int slot, int pat_num);
- Java: int get_pattern_y(int slot, int pat_num);
- JS: sv_get_pattern_y(slot, pat_num);
- Pixilang: sv_get_pattern_y(sv, pat_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- pat_num - pattern number.

Return value: Y coordinate of the pattern on the timeline.

sv_get_pattern_tracks()

Get the number of tracks of the specified pattern.

Prototypes:

- C: int sv_get_pattern_tracks(int slot, int pat_num);
- Java: int get_pattern_tracks(int slot, int pat_num);
- JS: sv_get_pattern_tracks(slot, pat_num);
- Pixilang: sv_get_pattern_tracks(sv, pat_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- pat_num - pattern number.

Return value: number of tracks or negative error code.

sv_get_pattern_lines()

Get the number of lines of the specified pattern.

Prototypes:

- C: int sv_get_pattern_lines(int slot, int pat_num);
- Java: int get_pattern_lines(int slot, int pat_num);
- JS: sv_get_pattern_lines(slot, pat_num);
- Pixilang: sv_get_pattern_lines(sv, pat_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- pat_num - pattern number.

Return value: number of lines or negative error code.

sv_get_pattern_name()

Get the name of the specified pattern.

Prototypes:

- C: const char* sv_get_pattern_lines(int slot, int pat_num);
- Java: String get_pattern_lines(int slot, int pat_num);
- JS: sv_get_pattern_lines(slot, pat_num);
- Pixilang: sv_get_pattern_lines(sv, pat_num);

Parameters:

- slot / sv - slot number / SunVox object ID;
- pat_num - pattern number.

Return value: pattern name or NULL.
In Pixilang the returned string container must be removed manually.

sv_get_pattern_data()
sv_set_pattern_data()

Access the contents of the pattern (array of events: notes, effects, etc.).

Prototypes:

- C: sunvox_note* sv_get_pattern_data(int slot, int pat_num);
- Java:

    - byte[] get_pattern_data(int slot, int pat_num);
    - int set_pattern_data(int slot, int pat_num, byte[] pat_data);
- JS: sv_get_pattern_data(slot, pat_num); //return value: UInt8Array
- Pixilang: sv_get_pattern_data(sv, pat_num); //return value: INT8 container

Parameters:

- slot / sv - slot number / SunVox object ID;
- pat_num - pattern number;
- pat_data - byte array to be written to the pattern.

Return value: pointer to an array with the contents of the pattern or NULL.
In Pixilang the returned container must be removed manually.
In C, JS and Pixilang you can read and write this array.
In Java you have to use get_pattern_data() to read and set_pattern_data() to write the array.

Pattern is a byte array of size (tracks*lines*8) bytes.
Here is the byte order in the array:

```
offset | description
       | line 0; track 0;
0      | note (NN)
1      | velocity (VV)
2      | module number + 1 (MM); low byte
3      | module number + 1 (MM); high byte
4      | effect code (EE)
5      | controller number + 1 (CC)
6      | controller value or effect parameter (YY); low byte
7      | controller value or effect parameter (XX); high byte
       | line 0; track 1;
8      | note (NN)
...    | ...
```

In Java, JS and Pixilang you can access it in the following way:

```
pat = sv_get_pattern_data(...);
ptr = ( line_number * sv_get_pattern_tracks(...) + track_number ) * 8;
note = pat[ ptr ];
velocity = pat[ ptr + 1 ];
module = pat[ ptr + 2 ] + ( pat[ ptr + 3 ] << 8 );
effect = pat[ ptr + 4 ];
controller = pat[ ptr + 5 ];
parameter = pat[ ptr + 6 ] + ( pat[ ptr + 7 ] << 8 );
```

In C there is a more convenient way. sv_get_pattern_data() will return a pointer to an array of the following structures:

```
typedef struct
{
    uint8_t     note;     /* NN: 0 - nothing; 1..127 - note num; 128 - note off; 129, 130... - see NOTECMD_* defines */
    uint8_t     vel;      /* VV: Velocity 1..129; 0 - default */
    uint16_t    module;   /* MM: 0 - nothing; 1..65535 - module number + 1 */
    uint16_t    ctl;      /* 0xCCEE: CC: 1..127 - controller number + 1; EE - effect */
    uint16_t    ctl_val;  /* 0xXXYY: controller value or effect parameter */
} sunvox_note;
```

sv_set_pattern_event()
sv_get_pattern_event()

sv_set_pattern_event() - write the pattern event to the cell at the specified line and track. Only non-negative values will be written to the pattern.
sv_get_pattern_event() - read a pattern event at the specified line and track.

- C:
  - int sv_set_pattern_event(int slot, int pat, int track, int line, int nn, int vv, int mm, int ccee, int xxyy);
  - int sv_get_pattern_event(int slot, int pat, int track, int line, int column);
- Java:
  - int set_pattern_event(int slot, int pat, int track, int line, int nn, int vv, int mm, int ccee, int xxyy);
  - int get_pattern_event(int slot, int pat, int track, int line, int column);
- JS:
  - sv_set_pattern_event(slot, pat, track, line, nn, vv, mm, ccee, xxyy);
  - sv_get_pattern_event(slot, pat, track, line, column);
- Pixilang:
  - sv_set_pattern_event(sv, pat, track, line, nn, vv, mm, ccee, xxyy);
  - sv_get_pattern_event(sv, pat, track, line, column);

Parameters:

- slot / sv - slot number / SunVox object ID;
- pat - pattern number;
- track - track number;
- line - line number;
- nn - note;
- vv - velocity;
- mm - module number + 1;
- ccee - controller + effect code (0xCCEE);
- xxyy - controller value or effect parameter (0xXXYY);
- column (field) - column to read from:
  - 0 - note (NN);
  - 1 - velocity (VV);
  - 2 - module (MM);
  - 3 - controller number or effect (CCEE);
  - 4 - controller value or effect parameter (XXYY);

sv_set_pattern_event() return value: 0 (success) or negative error code.
sv_get_pattern_event() return value: value of the specified field or negative error code.

sv_pattern_mute()

Mute / unmute the specified pattern.
USE LOCK/UNLOCK!

Prototypes:

- C: int sv_pattern_mute(int slot, int pat_num, int mute);
- Java: int pattern_mute(int slot, int pat_num, int mute);
- JS: sv_pattern_mute(slot, pat_num, mute);
- Pixilang: sv_pattern_mute(sv, pat_num, mute);

Parameters:

- slot / sv - slot number / SunVox object ID;
- pat_num - pattern number;
- mute: 1 - mute; 0 - unmute.

Return value: previous state (1 - muted; 0 - unmuted) or negative error code.

Other

sv_get_ticks()
sv_get_ticks_per_second()

SunVox engine uses system-provided time space, measured in system ticks (don't confuse it with the project ticks). These ticks are required for parameters of functions such as sv_audio_callback() and sv_set_event_t().
Use sv_get_ticks() to get current tick counter (from 0 to 0xFFFFFFFF).
Use sv_get_ticks_per_second() to get the number of system ticks per second.

Prototypes:

- C:
  - uint32_t sv_get_ticks(void);
  - uint32_t sv_get_ticks_per_second(void);
- Java:
  - int get_ticks();
  - int get_ticks_per_second();
- JS:
  - int sv_get_ticks();
  - int sv_get_ticks_per_second();
- Pixilang: see get_ticks() and get_tps();

sv_get_log()

Get the latest messages from the log.

Prototypes:

- C: const char* sv_get_log(int size);
- Java: String get_log(int size);
- JS: sv_get_log(size);
- Pixilang: see get_system_log();

Parameters:

- size - max number of bytes to read.

Return value: pointer to the null-terminated string with the latest log messages.