==
T H E " U N - O F F I C I A L"
PLAYSTATION DEVELOPMENT FAQ

SOUND
CONFERENCE
==

Release v1.1
Last Updated: August 31, 1995

DISCLAIMER

This FAQ is to aid in informing the licensed game developer about the development environment provided by Sony Computer Entertainment.

The Development System Tool to which this manual relates is supplied pursuant to and subject to the terms of the Sony Playstation Licensed Developer Agreement.

This FAQ is intended for distribution to and use only by Sony PlayStation Licensed Developers in accordance with the Sony Playstation Licensed Developer Agreement. The information in thismanual is subject to change without notice.

The content of this manual is Confidential Information of Sony for the purposes of the Sony PlayStation Licensed Developer Agreement and otherwise.

TRADEMARK INFORMATION

PlayStation and Sony Computer Entertainment names and logos are trade names and/or trademarks and/or copyright artwork of Sony Corporation(or its subsidiaries).

All specific names included herein are trademarks and are so acknowledged: IBM, Microsoft, MS-DOS. Any trademarks not mentioned here are still hypothetically acknowledged.

COPYRIGHT NOTICE

[1.] SOUND

[1.1.]: Reverb Setup

[1.1.1.]: With "libsnd" you can set 8 types of reverb mode; is there a function you can use to find out the size of the reverb work area taken in the SPU sound buffer for each mode?

We do not provide a function to find the reverb work area size. The size of the reverb work area taken in the sound buffer is uniquely determined by the mode decision. This area is acquired and held with not dynamic variation.

The table below gives the amounts of work area consumed. These are given also under **SpuSetReverbModeParam** in the SPU library document "function.txt".

Table: Space occupied in sound buffer for each reverb mode

Mode		Hex	Decimal
SPU_REV_MODE_OFF	off	80 (*)	128(*)
SPU_REV_MODE_ROOM	room	26c0	9920
SPU_REV_MODE_STUDIO_A	studio (small)	1f40	8000
SPU_REV_MODE_STUDIO_B	studion (medium)	4840	18496
SPU_REV_MODE_STUDIO_C	studion (large)	6fe0	28640
SPU_REV_MODE_HALL	hall	ade0	44512
SPU_REV_MODE_SPACE	space echo	f6c0	63168
SPU_REV_MODE_ECHO	echo	18040	98368
SPU_REV_MODE_DELAY	delay	18040	98368
SPU_REV_MODE_PIPE	pipe echo	3c00	15360

(*) From an address setting standpoint, even if mode is 'off', 128 bytes are consumed if it was **SpuReserveReverbWorkArea(SpuOn)** [that was called].
In the case of **SpuReserveReverbWorkArea(SpuOff)**, the space consumed is 0, the same as for other modes.

Related functions: **SpuReserveReverbWorkArea**, **SpuSetReverbModeParam**, **SpuSetReverb**

[1.1.2.]: What are the relationships between the work areas whose sizes vary according to the reverb mode setting, and the sound buffer memory management mechanisms ('SpuMalloc', etc.)?

[The work areas] are managed via the following algorithms.

1. Cases in which reverb work area has been reserved by means of **SpuReserveReverbWorkArea(SpuOn)** **SpuMalloc/SpuMallocWithStartAddr**
   Areas can be acquired in an area of [size] (0x7ffff - work area size), starting from address 0x01000, according to the mode.

2. Cases in which work area has been unreserved by means of **SpuReserveReverbWorkArea(SpuOff)** **SpuMalloc/SpuMallocWithStartAddr**
   Areas can be acquired in the entire sound buffer area, addresses 0x01000 to 0x7ffff.**SpuSetReverb**
   If an area with a size corresponding to the mode being used has been acquired as reverb work area in another area by means of **SpuMalloc/SpuMallocWithStartAddr**, then **SpuSetReverb(SpuOn)** will be ineffective [? or perhaps, "invalid" ?].

3. Regardless of the [?current?] reverb work area reservation, when a change is to be made to the reverb mode, **SpuSetReverbModeParam** analyzes whether or not [it] can acquire the area needed as work area, based on the information from the sound buffer memory management mechanisms, and sets the various attributes at that time if [that area] can be acquired. If [that area] cannot be acquired, it returns without setting [the attributes].

[? Translator's note: Original text here is syntactically ambiguous; above interpretation relies on technical context to resolve the ambiguities. ?]

4. If you execute **SpuMalloc/SpuMallocWithStartAddr** in a condition when there is no reverb work area reserved by **SpuReserveReverbWorkArea**, and afterwards attempt to reserve reverb work area anew by means of **SpuReserveReverbWorkArea**, then [it] analyzes whether or not [it] can acquire a reverb work area region of the size needed by the current reverb mode, based on the information from the sound buffer memory management mechanisms, and reserves that region at that time if [that area] can be acquired. If [that area] cannot be acquired, it returns without reserving [any work area].

5. The reverb work area [size] varies according to the reverb mode. The only time that the reverb work area size changes is when you set the mode with **SpuSetReverbModeParam**. The behavior of **SpuMalloc/SpuMallocWithStartAddr**, **SpuReserveReverbWorkArea**, and **SpuSetReverb** changes when the mode setting changes.

Related functions: **SpuReserveReverbWorkArea, SpuIsReverbWorkAreaReserved, SpuSetReverbModeParam, SpuSetReverb, SpuMalloc, SpuMallocWithStartAddr, SpuSetTransferStartAddr**

[1.1.3.]: Why doesn't reverb take effect even when I've turned reverb on and set the reverb parameters?

If you change the mode when setting reverb [parameters], noise may be generated for an instant due to invalid values remaining in the reverb work area.

In order to avoid this to the greatest degree possible, **SpuSetReverb(SpuOff)** is internally executed whenever the mode is changed. Since SPU_REV_MODE_OFF is set as the initial value of reverb mode, this means that even at the time of the first setting, reverb will end up being turned off by **SpuSetReverbModeParam**, even if you have previously executed **SpuSetReverb**. Therefore, you should always execute **SpuSetReverb** after **SpuSetReverbModeParam**.

Also, even if you execute **SpuSetReverb** immediately after executing **SpuSetReverbModeParam**, noise will inevitably be generated due to the inherent nature of reverb. To avoid this, you must obey the following rules until you truly need reverb:

- do not execute **SpuSetReverb(SpuOn)**
- do not set Depth in **SpuSetReverbModeParam**

Summarizing from the above, the points where care is required are:

1. If you are going to use reverb, then do the setup at the very start, not just before use. When you set it up, place it in the off state.

2. Do the reverb setup in the following order:
   SpuSetReverbModeParam
   SpuSetReverb

3. To avoid noise if you need to change the reverb mode during the game, avoid doing a set mode operation immediately before turning reverb on.

Note also that if you end up taking the entire sound buffer area for waveform data it will be impossible to acquire reverb work area, and therefore impossible to use reverb, since **SpuSetReverbModeParam** will end without setting the reverb mode, as described in "[3-2]:3.". To avoid this, keep track of your reverb work area sizes and waveform

data sizes, and set up the reverb modes and build the waveform data so that the required work area and waveform data will fit into 512 KB.

Related functions: SpuSetReverbModeParam, SpuSetReverb, SpuReserveReverbWorkArea, SpuIsReverbWorkAreaReserved, SpuSetTransferStartAddr, SpuMalloc, SpuMallocWithStartAddr

[1.1.4.]: Why doesn't reverb take effect even though I've set up the reverb parametersand turned reverb on as in [3-3]?

Because in the SoundDelicatessen (DTL-S710) Mac artist tool, the individual tones were not given a reverb attribute. Display the ADSR window with Program menu Æ Tone menu Æ ADSR ... , and set an [¥] in the 'reverb' check box in 'play mode' in that window. This will set the reverb attribute in the selected Tone's mode (bit 3 of **mode** = 1, i.e., **mode** = 4), so that reverb will take effect. Note, however, that reverb goes into effect universally on a Tone for which you set that check box to [¥], you must pay due consideration to reverb depth, overall atmosphere, and such when doing so.

[1.1.5.]: With programs which set reverb on, re-running the program sometimes generates noise. What about this?

When exiting a program that uses reverb, you absolutely must do the following:

∑ SPU library case:

```
#include <libspu.h>

SpuReverbAttr r_attr;
r_attr.mask        = (SPU_REV_MODE |
                   SPU_REV_DEPTHL | SPU_REV_DEPTHR);
r_attr.mode        = SPU_REV_MODE_OFF;
r_attr.depth.left  = 0;
r_attr.depth.right = 0;

SpuSetReverbModeParam (&r_attr);
SpuSetReverb (SpuOff);
```

∑ Sound library case:

```
#include <libsnd.h>

SsUtReverbOff();
SsUtSetReverbType (0);
SsUtSetReverbDepth (0, 0);
```

If you do not do this, you may sometimes get noise on the next run.

Related functions: **SpuSetReverbModeParam**, **SpuSetReverb**, **SsUtReverbOff**, **SsUtSetReverbType**, **SsUtSetReverbDepth**

[1.2.]: Sound Volume

[1.2.1.]: There is a sound volume setting function CdMix in libcd. Is this used to set the CD/DA and/or CD-ROM/XA sound volume?

In the case of CD output, whenever the CD decoder interprets what is read from the CD as music data such as CD/DA or CD-ROM/XA (ADPCM) [data], it splits that data into **left** and **right** and sends it to the SPU, where it is input to the SPU section called **Serial A** and mixed with the sound data output by the SPU, and then output from the audio output. The SPU library and sound library (**libsnd**) provide functions to manipulate the sound volume of this music data from the CD. The functions and their usage are given below.

∑ SPU library

```
#include <libspu.h>

SpuCommonAttr attr;

attr.mask = (SPU_COMMON_MVOLL  | /* master volume (left)   */
             SPU_COMMON_MVOLR  | /* master volume (right)   */
             SPU_COMMON_CDVOLL | /* CD input volume (left)  */
             SPU_COMMON_CDVOLR | /* CD input volume (right)  */
             SPU_COMMON_CDMIX);  /* CD input on /off  */

/* set master volume to mid-range */
attr.mvol.left  = 0x1fff;
attr.mvol.right = 0x1fff;

/* set CD input volume to mid-range */
attr.cd.volume.left  = 0x1fff;
attr.cd.volume.right = 0x1fff;

/* CD input ON  */
attr.cd.mix = SpuOn;

/* set attributes */
SpuSetCommonAttr (&attr);
```

∑ Sound library (**libsnd**)

```
#include <libsnd.h>

/* CD input  ON */
```

```
SsSetSerialAttr (SS_SERIAL_A, SS_MIX, SS_SON);

/* set volume to mid-range */
SsSetSerialVol (SS_SERIAL_A, 0x40, 0x40);
```

You can manipulate the volume of music data from CD via either of the above setups.

Related functions: **SpuSetCommonAttr**, **SsSetSerialAttr**, **SsSetSerialVol**

[1.2.2.]: Whenever I execute SpuInit (or SsInit) after executing CdInit, I lose the CD sound that I had been getting up to then by executing CdInit alone. What about this?

Both **SpuInit** and **SsInit** set the CD volume to 0. This is because, just as their names indicate, they are for sound system initialization.

Therefore, to use music data from the CD, you absolutely must use **SpuSetCommonAttr** (in the sound library, **SsSetSerialAttr**, and **SsSetSerialVol**) as described in B-1.

Related functions: **SpuSetCommonAttr**, **SsSetSerialAttr**, **SsSetSerialVol**

[1.3.]: Finding Out The Key Status Of An Individual Voice

[1.3.1.]: In cases with libspu where I'm using 24 voices, is there a way that I can find out whether a particular voice is currently keyed on?

By using **SpuGetKeyStatus** and **SpuGetAllKeysStatus** you can check the key status for a particular voice or for all voices.

Related functions: **SpuGetKeyStatus**, **SpuGetAllKeysStatus**

[1.4.]: Voice Data Attributes

[1.4.1.]: I want to force looping on the waveform of a voice that I've keyed on with SpuSetKey. But it won't loop for me even though I set SPU_VOICE_LSAX in mask in the SpuVoiceAttr

structure and call SpuSetVoiceAttr **specifying the address in the sound buffer in** loop_addr**.**

The only waveform data for which you can modify looping with **SPU_VOICE_LSAX** is that waveform data which was set up for looping at the time it was created. If looping wasn't set up when it was created, setting **SPU_VOICE_LSAX** and calling **SpuSetVoiceAttr** will have no effect. If you want to do looping, you must set up the looping when you create that waveform data.

Related functions: **SpuSetVoiceAttr**, **SpuSetKey**

[1.5.]: Waveform Transfer

[1.5.1.]: If I transfer data to the sound buffer by means of I/O transfer, do I need to call SpuIsTransferCompleted to check for completion?

You do not need to use **SpuIsTransferCompleted** to check whether an I/O transfer has completed, because execution will return from **SpuWrite** when processing is complete. If you call **SpuIsTransferCompleted** when the transfer mode is I/O transfer, **SpuIsTransferCompleted** will return 1 regardless of its argument.

Related functions: SpuIsTransferCompleted, SpuWrite, SpuRead, SpuSetTransferMode

[1.5.2.]: Why doesn't execution return from SpuIsTransferCompleted if I call it to check for completion after executing a DMA transfer?

The overall operation of the current library is such that you must execute either **_96_remove()** or **CdInit()** ahead of all other processing. If you do not, execution may not return, especially for S**puIsTransferCompleted(SPU_TRANSFER_WAIT).**

Related functions: **SpuIsTransferCompleted**, **SpuWrite**, **SpuRead**, **SpuSetTransferMode**, **_96_remove**

[1.6.] Miscellaneous

[1.6.1.]: How many VAB data can be opened?

Up to 16 VAB data can be opened simultaneously. When opening some data, repeat the following procedure.

SsVabOpenHead() --> SsVabTransBody() --> SsVabTransCompleted()

[1.6.2.]: What in the VAB file corresponds to the MIDI data track number?

The MIDI data track number corresponds to the VAB data program number. It can be said that the "program" of VAB data is equivalent to the program set by the MIDI program change.

[1.6.3.]: How should the VAB data program be handled in order to play a track with a MIDI instrument?

For example, if the forth MIDI track is played by drums, put the drum sound into the forth VAB data program.

1.6.4.]: How should an instrument be set if it is switched to another in the same track during a piece of music?

Each instrument should be prepared as the separated program in the VAB data because the MIDI program change is used for switching the instrument in the track.

[1.6.5.]: Repeating ON/OFF for the reverb doesn't succeed.

What is the cause?

The following will cause the improper reverb operation. Please check them.

1) Since the SPU local memory is used up, the reverb work area cannot be allocated.

2) The reverb utilization attribute (mode) is not set properly in Sound Delicatessen.

[1.6.6.]: How can the loop of music (such as endless playback of a piece of music) be performed?

The loop of music can be performed by the SsSeqPlay() function as follows:

1) A piece of music is played back 3 times.
```
SsSeqPlay({SEQ address}, SSPLAY_PLAY, 3);
```

2) A piece of music is played back endlessly.
```
SsSeqPlay({SEQ address}, SSPLAY_PLAY,          SSPLAY_INFINITY);
```

[1.6.7.]: Can the VAB data read in the parent process be used as it is when executing the child process with Exec()?

Under the current library specification, the internal status will be cleared when the Exec()function is executed. Thus, VAB and SEQ must be set and transferred every time in the child process.

[1.6.8.]: How can stereo/monaural modes at CD-DA playback be switched?

Use the CdMix() function to switch the modes. The SsSetStereo() and SsSetMono() functions are provided for the SPU sound.

[1.6.9.]: Why does the SsVabOpen() function not work after the SsEnd() execution?

It is the proper operation. Since the SsEnd() function stops the sound system, the SsVabOpen() function doesn't work after the SsEnd() execution. The SsEnd() function must be used for stopping the sound system
in such a case as performing the overlay.

[1.6.10.]: Why does the sound of moving pictures not come out when executing the SsInit() function after the execution of CdInit()?

Because, while the CdInit() function turn up the volume, the SsInit() function initializes all the volumes. However, the order of initialization
(CdInit() ---> SsInit()) is correct. Thus, after the initialization, set the CD mixing and volume effective
with the SsSetSerialAttr() and SsSetSerialVol() function at the necessary timing.

[1.6.11.]: Why is '-1' returned at the second SsVabOpen()when calling the function in a row?

Since the SsVabOpen() function performs the wave form data transfer into the sound buffer, it is necessary to check the end of the transfer with the SsVabTransCompleted() function after calling the SsVabOpen() function. Without calling SsVabTransCompleted(), the second SsVabOpen() will be rejected by the library, and '-1' will be returned. Moreover, the wave form data in VAB data will be left in main memory. For the effective
use of memory, using the SsVabOpenHead() and SsVabTransBody() functions is recommended.

[1.6.12.]: Which reverb type is effective, set with NRPN control or set by the SsUtSetReverbType() function?

It is not until 'Reverb Type' data set with NRPN is recognized that it becomes effective instead of 'Reverb Type' set by the sound utility functions. Other 'Reverb' information will be treated in the same way.

[1.6.13.]: When describing NRPN which sets the reverb without the DSP program designation in a program and playing it, do the Ss- functions set the DSP program of 'Room' type?

As for the reverb settings in NRPN, such attributes as 'Mode', 'Reverb Type', and 'Reverb Depth', must be set each. Moreover, when describing NRPN which sets the reverb without the DSP program designation in the program and playing it, the settings will be performed by setting the reverb ON and setting 'Depth' with the sound utility functions. However, if 'Depth' setting is performed before 'Type' setting, noise may be generated by the remarkable change in the volume.
Thus, it will be better to perform all the settings with NRPN.

[1.6.14.]: Are there any differences among the followings?

*** Reverb depth controlled by NRPN * 'external effect depth' specified by the control number #91 * Reverb depth set by the SsUtSetReverbDepth() function**

No. All is the same.

[1.6.15.]: Does the start point of a loop indicate the location described by NRPN, or the location where the immediately next MIDI event exists?

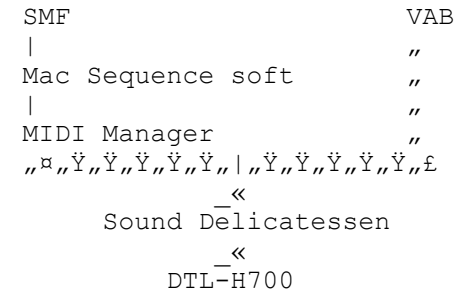
The start point of the loop becomes effective from the MIDI event immediately after the NRPN description. That is to say, the MIDI data after the data1 description of Loop1 becomes the start pointer.

[1.6.16.]: Is it enough for the SEQ data with the endless-loop description with NRPN to be played only once?

Yes. Once is enough as for the SEQ data with the endless-loop setting.

[1.6.17.]:Why does the loop designated with Control Change in NRPN not work with Sound Delicatessen?

As for the SEQ playback, Sound Delicatessen doesn't correspond to NRPN. In addition, since the SEQ playback with Sound Delicatessen has such problems as the unstable tempo, it could not be recommended. Check the conformity between the SEQ data and the VAB data with a sequencer software on Macintosh via MIDI Manager or Free MIDI. Since DTL-H700 can be recognized as the virtual MIDI instrument by MIDI Manager with Sound Delicatessen, SMF and VAB will be played according to the following structure.



However, the current NRPN version corresponds to the reverb only even if using MIDI Manager. Because, for example, Sound Delicatessen cannot control the loop in the sequence soft. Therefore, in order to check the other NRPNs such as the loop, use Programmer Tool.

[1.6.18.]:Is there any certain ways to generate the sound of a specific instrument in libsnd?

Yes. Increase the used VAB program priority in ToneList.

[1.6.19.]:Is there any ways to reserve some channels for the sound effect?

Calling the SsSetReservedVoice() function immediately after SsInit() will limit the number of voices allocated by libsnd. The remaining channels can be used for the sound effect in libspu.

[1.6.20.]:What should be set for 'ATTR' on 'Programs' screen of Sound Delicatessen?

ATTR(attribute) is not used by the sound library.
16-bit short variables can be set here, and values from -32767 to 32768 are valid. This can be used freely.

[1.6.21.]:What is the priority setting on 'Tones' screen of Sound Delicatessen?

The priority is provided for determining which sound should be turned off prior to others. It can be set at 0 to 127, and the higher the priority becomes, the later the sound will be turned off. For example, in the case of a lot of simultaneous sound generation, in order to be sure to generate the melody line and the drum part, the priorities of the melody and the tone used by the drum must be higher.

- ----------

[1.6.22.]:Can some VAB data be contained in the sound buffer and used as separate sound sources simultaneously? For example, VAB1 is assigned for SEQ, and VAB2 for other sound effects.

The VAB-related processing must be performed twice to do it. Refer to the following sample.

/*
For convenience's sake, the printf() function is used for the error output. For example, vab1 must be used as VAB ID when using SEQ, and vab2 must be used as VAB ID when using the sound effect. However, if the total of VB capacity is over the sound buffer capacity,
it cannot be used. The sound buffer capacity is as follows:

[512KB - 0x1010 - (Capacity of the reverb work area)] bytes.

```
Unless the reverb work area is used, 80 bytes will be consumed with the
current library.
*/

short vab1, vab2;

/* ----------------------------------------------------------------
* VAB #1 transmission
* ---------------------------------------------------------------- */

/* VH #1 transmission */
vab1 = SsVabOpenHead ((u_char *)VH1_ADDR, -1);
if (vab1 == -1) {
        printf (""SsVabOpenHead : Can NOT open header.\n"");
        return;
}

/* VB #1 transmission */
if (SsVabTransBody ((unsigned char *) VB1_ADDR, vab1) != vab1) {
        printf (""SsVabTransBody : failed !!!\n"");
        return;
}
SsVabTransCompleted (SS_WAIT_COMPLETED);

/* ----------------------------------------------------------------
* VAB #2 transmission
* ---------------------------------------------------------------- */

/* VH #2 transmission */
vab2 = SsVabOpenHead ((u_char *)VH2_ADDR, -1);
if (vab2 == -1) {
        printf (""SsVabOpenHead : Can NOT open header.\n"");
        return;
}

/* VB #2 transmission */
if (SsVabTransBody ((unsigned char *) VB2_ADDR, vab2) != vab2) {
        printf (""SsVabTransBody : failed !!!\n"");
        return;
}
SsVabTransCompleted (SS_WAIT_COMPLETED);
```