

# PlayStation Sampler Disk Specification for Playable Game Segments

## Overview

This document sets out the guidelines for generating a playable game demonstration to be included in a PlayStation sampler. This document includes a specification of the details involved in generating a demonstration, and a technical checklist.

## Technical Specification

### Launcher Program

This directory contains launcher, a simple launch program based on the one for the full demo disk. There is a makefile for making launch.exe, the source for launch, in case you need to make temporary changes (temporary, mind) and an example cti file showing how to lay out an example program on the emulator to be loaded by and run by launch. Incidentally, launch uses bs.lib, a specially built subset of the full libraries. bs.lib is copyright SCEE.

### What does launch do ?

Basically, it boots the machine, displays a crappy icon, and then loads and runs the executable at the location specified in source. Currently, this location is sector 24. You will likely need to modify this location to match your situation.

If your code is all ok, and all is well, the dev system should printf to the host:

```
<Some stuff about the bs heap, data locations and so on>  
<Followed by the stuff from PadInit(>  
<and lastly:>
```

Launcher: Set loc for seek

Launcher: Program go.

At this point, launch has seeked to your program and is loading it. Immediately after this, your program will be run. If something goes wrong, launch will timeout and reboot the machine. This will probably cause repeated reboots, but you never know. Now your game should be running.

The executable will be loaded according to the XF\_HDR data at the start of the .EXE, i.e., it should have correct info in it or all will be lost; the program is loaded to the address specified in the t\_addr field, and the bss is cleared.

## Overview

The way the demo disk actually works is as follows. When the PlayStation boots up at power on, the launcher program is loaded from CD into main RAM and runs. The launcher is loaded to and runs within the 32K bytes of RAM right above the PlayStation kernel's 64K of RAM (80010000 - 80018000). The launcher then launches the menu program, which allows the user to select and play the various games, and do whatever other activities are provided. Once the user has chosen your playable demo, your executable (.EXE) will be loaded from CD; the BSS segment will be cleared, and your program will be executed. While your program is running, the launcher will still be in RAM, so your program \_must\_ not write to memory between 0x80001000 and 0x80018000. Since

you are probably used to not touching memory inside the kernel's space, it is anticipated that altering your code to avoid corrupting the launcher will probably only require that you re-link your code with an org address 32K bytes higher than before. Hopefully, losing this 32K of RAM will not require a lot of changes to your game code. If you are pushed for RAM, you may consider checking the size of the stack you are using - the default is 32K, which is pretty big. The launcher keeps its own small stack inside its 32K, so you don't need to worry about corrupting its stack; your stack (in standard configuration) is in the top 32K of memory. With the launcher loaded, and your program running, the main RAM looks like:

```
0x80000000 - 0x80010000 PlayStation Kernel RAM space
0x80010000 - 0x80018000 launcher program and stack
0x80018000 - 0x80....   Your demo code and data
0x801f8000 - 0x801fffff Your stack (assuming the default size and position
                        assigned by libsn)
```

In order to maintain sanity, your demo must live within its own directory on the CD, which will contain the data files used by your demo and any other information it needs, with the exception of any DA audio, which will be a separate track on the CD. You are encouraged to keep the number of files you use to a minimum, as some other pieces of code on the disk may use CdSearchFile, with its limitations of around 40 directories with about 30 files in each. If your application uses a number of files anywhere near this limit, then your application will likely cause other applications to fail (in which event, the applications are prioritized). The launcher runs your program using the kernel call Exec(), and so your playable demo must be a standard PlayStation .EXE file. Critically, in order for your playable demo to accept arguments, and return control to the launcher properly, you must link your code with the provided startup.obj, a replacement for the startup code in libsn.lib. This start-up module does not clear the bss and set up the heap, because if your playable demo does this, it may overwrite the launcher. The only problem is that code linked with startup.obj will not run in its own right (obviously). So that you can test your executable, a simplified version of our application launcher is provided with this distribution.

In addition to the memory and start-up restrictions described above, your program must do its initialisation, and tear down, as below. This small fragment of code is essentially a harness for a program that will return properly to the launcher and also leave the various PlayStation subsystems in a usable state.

```
#ifdef LINKED_STARTUP           /* If we have linked startup.obj */
int main( int argc, char** argv) /* launcher will pass argc, argv to you. */
#else
int main()                      /* Plain old main instead. */
#endif                          /* PSX doesn't like argc, argv in a main
                                prog */

{
    ResetCallback();            /* Clear all of the CD callbacks. */
    CdInit();                   /* Re-initialise the CD subsystem. */
    PadInit(0);                 /* Initialise the pads. */
    ResetGraph(0);              /* Cold boot the GPU. */
    SetGraphDebug(0);           /* Turn GPU debugging off. */

    /* Now you can do any other startup you need to */
    /* And the code from here on is your own          */
```

```

        .
        .
        .
/* Little Johnny has been killed by the giant spiders from Mars, so.. */
/* This is the end of the program now. */

/* reverse any callback initializations */
/* stop and close any events */
/* failure to do these almost reliably causes crashes in the launcher or in
other demos, and will equally reliably involve another revision of your demo
*/

StopCallback();      /* Stop the CD callbacks. */
PadStop();           /* Stop pad reading. */
ResetGraph(3);       /* GPU warm reset */
return (0);          /* This is necessary too. */
}

```

The launcher will pass the standard C variables argc and argv to your program. However, argv will not be a ragged array of characters, it will actually just point to an array of integers; argc will be four, as your demo will be passed four pieces of information. The contents of argv will be: the mode of the demo (interactive or non-interactive), the time-out you will use to stop the playable demo, the sector # for the directory containing your app, and the track index of the game's first DA track. We will ensure that your DA tracks are stored sequentially beginning at the specified track number. DA track numbers are 1 indexed. If the game does not use DA, arguments four will be zero and can be ignored. While you are free to use whatever mechanism you choose to locate your files on the cd, the sector # for your application's directory is provided for speed, as your file positions can be hard coded as offsets from this value.

You can figure out the mode, time-out, position, and DA information using the following code:

```

#define INTERACTIVE          0
#define NON_INTERACTIVE     1

#ifdef LINKED_STARTUP        /* If we have linked startup.obj */
int main( int argc, int* argv) /* launcher will pass argc, argv to you.
                               */
                               /* argv is actually an array of integers.
                               */
#else
int main()                  /* Plain old main instead. */
#endif                      /* PSX doesn't like argc, argv on its own
                               */
{
    int mode;                /* Demo mode */
    int timeout;              /* timeout in seconds. */
    int first_DA_track;      /* First DA track */
    int sector_offset;       /* location of application directory */

    /* All the usual startup stuff etc. */

#ifdef LINKED_STARTUP
mode = argv[0];

```

```
timeout = argv[1];
first_DA_track = argv[2];
sector_offset = argv[3];
#endif

/* And on with the action */
```

Submission Form  
(submit with demonstration)

Game Title:  
Publisher:  
Developer:  
Project Supervisor (Name, Phone, E-mail):  
  
Primary Technical Contact (Name, Phone, E-mail):

Style of Game (e.g. shoot-em-up, racing, etc.):

Game Function: Playable? Auto-Play for Non-Interactive? FMV only?

Controller / Peripheral Support (final game & this demo):

Size of EXE on CD (bytes):  
Size of Data on CD (bytes):

Build Instructions (alternatively, submit a .cti or .ccs file with your data):

Playable Section Technical Checklist  
(MUST BE SUBMITTED WITH DEMONSTRATION)

Please verify all of the following technical details before submitting your demonstration:

- o Demo linked with sampler.obj.
- o Demo orged at 0x80018000 or above.
- o Demo does not write to memory in range 0x80010000 r 0x80018000.
- o Demo is configured for NTSC.
- o Demo does startup and shutdown as specified above.
- o Demo behaves according to the specified mode (interactive/noninteractive).
- o Demo programs using DA pay attention to argv[3] in order to determine the appropriate DA tracks.
- o Demo executable and data lives within its own directory.
- o Demo can be quit in either mode with 'select' key at any point, on any screen.
- o Demo falls out of main in order to exit.
- o Demo terminates at the end of a game segment or after the specified timeout.
- o Interactive demo will never sit without input for more than the specified timeout.
- o Demo removes any callbacks and event handlers before closing down.
- o Demo is not in the middle of DMA when closing down.
- o Demo clears the reverb buffer in SPU RAM before closing down.

- o On a debug station, demo is tested to indefinitely launch successfully from the provided launcher, and return control to the launcher. (This loop should be verified for an absolute minimum of 10 iterations in both interactive and noninteractive modes).
- o Demo does not depend on the state or contents of RAM (other than BSS), data cache, VRAM, SPU RAM, CD sector buffer, or I cache at startup.
- o Stack pointer has been set properly in your .EXE header to the value that you require (by you - use setsp program included with harness distribution).
- o Printf() inserted at the start of main() showing arguments received by the launcher.
- o Printf() inserted at the end of main() to mark end of demo execution.