Video Player Library Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview 3
Characteristics 3
Files 3
Sample Programs 3
Reference Materials 3
2 Using the Library 4
Basic Procedure 4
List of Functions 7
3 Notes 8
Jumping in Pause Mode 8

1 Library Overview

Characteristics

The video player library is a high-level library for playing back MP4 files containing AVC video and AAC audio streams.

Files

The files required to use the video player library are shown in Table 1.

Table 1 Required Files

File Name	Description
`sceavplayer.h`	Header file
`libSceAvPlayer_stub.a`	Stub for library

Sample Programs

The sample programs for the video player library are as follows:

`samples/sample_code/audio_video/api_libavplayer/simple_mp4_player`

This sample program demonstrates the basic use of the video playback library.

`samples/sample_code/audio_video/api_libavplayer/advanced_mp4_player`

This sample program demonstrates the high-level use of the video playback library.

Reference Materials

For details about the various functions, refer to the *Video Player Library Reference* document.

2 Using the Library

Basic Procedure

Source Content

Standard MP4 files containing AVC Video and AAC Audio are supported. You can use a number of commercial encoding tools, such as Sony Vegas, to create valid streams.

To ensure optimal playback performance, you should optimize the video for streaming. This is also referred to as having the "MOOV atom at the front" and being "tightly interleaved". There are a number of free tools available on the internet that can remux/reinterleave MP4 files so playback is of optimal quality.

Load / Unload sceavplayer Module

Before calling any video player library functions, you must load the sceavplayer module (`SCE_SYSMODULE_AV_PLAYER`). You must also ensure that you have stopped playback and closed the library before unloading the module.

Initialize Library

To perform initialization, call `sceAvPlayerInit()`.

- Pass the initialization parameters for the memory allocators and provide file access and event callbacks.
- The sample includes example replacement memory and file calls in `replacement.h`. You can use them as-is inside of your game if you don't need to customize them.
- Additional initialization parameters:
  - **debugLevel** will output additional information to the console which should help when debugging crashes or file compatibility issues. To ensure no unnecessary text is output to the console in your final build, make sure that the *`debugLevel`* is set to `SCE_AVPLAYER_DBG_NONE` in your final title.
  - **basePriority** accepts a value from 125 to 175. Ensure that your base priority is greater than your rendering thread; otherwise the player may behave in an unexpected manner.
  - **numOutputVideoFrameBuffers** can be adjusted to compensate for rendering latency. 2 or 3 should generally always work.
  - **autoStart** allows the player to start without having to call `sceAvPlayerEnableStream()` and `sceAvPlayerStart()` from the supplied callback. Note that when *`autoStart`* is true, you can set *`eventCallback`* to NULL, and you do not have to set up the streams that you wish to play.
  - **defaultLanguage** is only used when *`autoStart`* is set to true. This allows the *`autostart`* to select the correct audio or video track when supplied with a stream that contains multiple languages.
- Note that overriding the file access is not a necessity. However, you do have to override the memory access.

Example:

```
SceAvPlayerInitData playerInit;
playerInit.memoryReplacement.objectPointer = NULL;
playerInit.memoryReplacement.allocate = Allocate;
playerInit.memoryReplacement.deallocate = Deallocate;
playerInit.memoryReplacement.allocateTexture= AllocateTexture;
playerInit.memoryReplacement.deallocateTexture = DeallocateTexture;

playerInit.fileReplacement.objectPointer = NULL;
playerInit.fileReplacement.open = OpenFile;
playerInit.fileReplacement.close = CloseFile;
```

```
    playerInit.fileReplacement.seek = SeekFile;
    playerInit.fileReplacement.readOffset = ReadOffsetFile;
    playerInit.fileReplacement.size = SizeFile;

    playerInit.eventReplacement.objectPointer = NULL;
    playerInit.eventReplacement.eventCallback = EventCallback;

    playerInit.debugLevel = SCE_AVPLAYER_DBG_ERRORS_ONLY;
    playerInit.basePriority = 160;
    playerInit.numOutputVideoFrameBuffers = 2;
    playerInit.autoStart = false;
    playerInit.defaultLanguage = "eng";

    g_samplePlayer = sceAvPlayerInit(&playerInit);
```

Set Video File

This is an asynchronous call. You can complete initialization of the stream directly from the event callback supplied in the previous call to `sceAvPlayerInit()`.

Example:

```
    sourceID = sceAvPlayerAddSource(g_samplePlayer, FILENAME);
```

Enable Streams

Changes in the player state trigger a call to the supplied *`eventCallback`*. As soon as the player has reached the `SCE_AVPLAYER_STATE_READY` state, you can use the event callback to safely retrieve the number of streams that are available and then enable them. Generally, you will want to do this directly from the callback or, alternatively, use the callback to alert the main thread that the streams are available and ready to be enabled.

Note: This is not required if you initialize the player with *`autoStart`* set to true.

Example:

```
    streamCount = sceAvPlayerStreamCount(g_samplePlayer);

    for (int i = 0; i < streamCount; i++) {
        sceAvPlayerStreamInfo StreamInfo;
        ret = sceAvPlayerGetStreamInfo(g_samplePlayer, i, &StreamInfo);

        if (StreamInfo.type == SCE_AVPLAYER_VIDEO) {
             ret = sceAvPlayerEnableStream(g_samplePlayer, i);
        }
        else if (StreamInfo.type == SCE_AVPLAYER_AUDIO) {
            ret = sceAvPlayerEnableStream(g_samplePlayer, i);
        }
        else if (StreamInfo.type == SCE_AVPLAYER_TIMEDTEXT) {
            ret = sceAvPlayerEnableStream(g_samplePlayer, i);
        }

    }
```

Tip: If you get an error returned from `sceAvPlayerEnableStream()`, you should ensure that the graphics memory allocator supplied to the library is valid. For more information, see tech note https://psvita.scedev.net/technotes/view/103/1.

Start Player

This call can only be used after successfully enabling the streams you wish to play. Note that there needs to be at least one stream.

Note: This is not required if you initialize the player with *autoStart* set to true.

Example:

```
sceAvPlayerStart(g_samplePlayer);
```

Start Audio Output Thread

Each audio sample includes details that you can use to ensure that it is output correctly, which include the sampling rate and channel count. In your audio output thread, it is important to handle the scenario where audio is not delivered as well as when it is.

Example:

```
AudioOutThread()

{
    SceAvPlayerFrameInfo audioFrame;

    while(g_audioActive) {
        if (sceAvPlayerGetAudioData(g_samplePlayer, &audioFrame))
        {
            // Synchronous sound output here
        } else {
            // Synchronous sound output of an empty PCM buffer to prevent
            // player performance from being affected.
        }
    }

    sceKernelExitThread(0);
}
```

Draw Video Frames

Each video frame includes details that you can use to ensure that it is output correctly, including the resolution and aspect ratio.

Example:

```
while (sceAvPlayerIsActive(g_samplePlayer)) {
    if (sceAvPlayerGetVideoData(g_samplePlayer, &videoFrame)) {
        // Draw Frame
    }
    // vsync flip
}
```

Handling embedded timed text streams (subtitles)

Each text frame includes basic details that you can use to ensure that it is output correctly. The sample outputs to TTY. Games, however, can route the text output to the screen if they wish to have subtitles. The text streams are delivered at the point in time they are meant to be output, so the text needs to be displayed in the next possible video frame.

Example:

```
case SCE_AVPLAYER_TIMED_TEXT_DELIVERY:
    {
        SceAvPlayerFrameInfo* pTextEventData =
            reinterpret_cast<SceAvPlayerFrameInfo*> (argEventData);
        std::string textData((char*)pTextEventData->pData,
    pTextEventData->details.subs.textSize);
        printf("## advanced_mp4_player: Text to display:: %s \n",
    textData.c_str());
    }
break;
```

Termination

When you are finished with the library, the player should be closed. If you are closing a video before the end of the file without calling `sceAvPlayerStop()`, you should expect *`eventCallback`* to be triggered after this call is made.

Example:

```
SceAvPlayerClose(g_samplePlayer);
```

List of Functions

Table 2 Library Functions

libSceAvPlayer	Description
`SceAvPlayerHandle sceAvPlayerInit(SceAvPlayerInitData*);`	Initializes the player and supplies it with initialization parameters.
`int32_t sceAvPlayerPostInit(SceAvPlayerHandle h, SceAvPlayerPostInitData* pPostInit);`	Allows advanced initialization of the libsceAvPlayer API. Use of this function is optional, and it should be used with caution.
`int32_t sceAvPlayerClose(SceAvPlayerHandle h);`	Closes the player and frees any outstanding memory allocations.
`int32_t sceAvPlayerAddSource(SceAvPlayerHandle h, char* argFilename);`	Adds the source file to the player instance.
`int32_t sceAvPlayerStreamCount(SceAvPlayerHandle h);`	Retrieves how many valid streams are available.
`int32_t sceAvPlayerGetStreamInfo(SceAvPlayerHandle h, uint32_t argStreamID, SceAvPlayerStreamInfo* argInfo);`	Retrieves detailed information on each stream.
`int32_t sceAvPlayerEnableStream(SceAvPlayerHandle h, uint32_t argStreamID);`	Enables a stream.
`int32_t sceAvPlayerDisableStream(SceAvPlayerHandle h, uint32_t argStreamID);`	Disables a stream.
`int32_t sceAvPlayerStart(SceAvPlayerHandle h);`	Starts playback.
`int32_t sceAvPlayerSetLooping(SceAvPlayerHandle h, bool loopflag);`	Sets the looping mode.
`int32_t sceAvPlayerStop(SceAvPlayerHandle h);`	Stops playback.
`int32_t sceAvPlayerPause(SceAvPlayerHandle h);`	Pauses playback.
`int32_t sceAvPlayerResume(SceAvPlayerHandle h);`	Resumes playback from pause.
`int32_t sceAvPlayerGetVideoData(SceAvPlayerHandle h, SceAvPlayerFrameInfo*);`	Retrieves the relevant video frame.
`int32_t sceAvPlayerGetAudioData(SceAvPlayerHandle h, SceAvPlayerFrameInfo*);`	Retrieves the relevant audio frame.
`bool sceAvPlayerIsActive(SceAvPlayerHandle h);`	Checks the status of player.
`uint64_t sceAvPlayerCurrentTime(SceAvPlayerHandle h);`	Retrieves the current playback time.
`bool sceAvPlayerJumpToTime(SceAvPlayerHandle h, uint64_t argOffsetMsec);`	Jumps to a time offset in ms.
`bool sceAvPlayerSetTrickSpeed(SceAvPlayerHandle h, int32_t argTrickSpeed);`	Sets the trick mode speed
`bool sceAvPlayerGetVideoDataEx(SceAvPlayerHandle h, SceAvPlayerFrameInfoEx* videoInfo);`	Retrieves the relevant video frame and the extended details about it.

3 Notes

Jumping in Pause Mode

While jumping in pause mode, you should ensure that video frames are requested; otherwise consecutive jump requests will fail. A single video frame will be available as output for every jump request made. This is useful for tasks such as timeline thumbnail generation.