libatrac Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview

Scope of This Document

This document describes the libatrac library, which is a streaming library for ATRAC™. libatrac can be used to easily implement streaming playback of ATRAC9™ data and a loop sound source. In addition, libatrac can be used to change the playback position of ATRAC9™ data and play back data with an epilogue after the ATRAC9™ data loop.

Purpose and Features

libatrac is a library for controlling the ATRAC9™ decoder and the input buffer.

The application can use libatrac to easily implement streaming playback and playback of a loop sound source.

libatrac carries out decode processing by using libaudiodec.

Main Features

libatrac provides the following main features.

- Streaming playback of ATRAC9™ data
- Loop playback of ATRAC9™ data in a sample unit
- Dynamic changes of the ATRAC9™ data playback position in a sample unit
- Playback of data with an epilogue after the ATRAC9™ data loop

Used Resources

libatrac uses the following system resources.

Resource	Description
Footprint	36 KiB when PRX is loaded
Work memory	Work memory (1 to 16 KiB) explicitly provided by the application
Thread	An internal thread is not created implicitly. The thread calling the API performs the operation.

Embedding in Program

The files required for using libatrac are as follows.

Filename	Description
atrac.h	Header file
libSceAtrac_stub.a	Stub library file
libSceAtrac_stub_weak.a	weak import stub library file

Include libatrac.h in the source program. Various header files will be automatically included as well.

Load the PRX module in the program, as follows.

```
if ( sceSysmoduleLoadModule(SCE_SYSMODULE_ATRAC) != SCE_OK ) {
    // Error handling
}
```

Upon building the program, link libSceAtrac_stub.a or libSceAtrac_stub_weak.a.

Sample Programs

The following file is provided as a sample program that uses libatrac.

sample_code/audio_video/api_libatrac/basic/

This sample shows the basic usage of the libatrac

Reference Materials

For audio output, refer to the following document.

- Audio Output Function Overview

For libaudiodec used by libatrac, refer to the following document.

- libaudiodec Overview

2 Using the Library

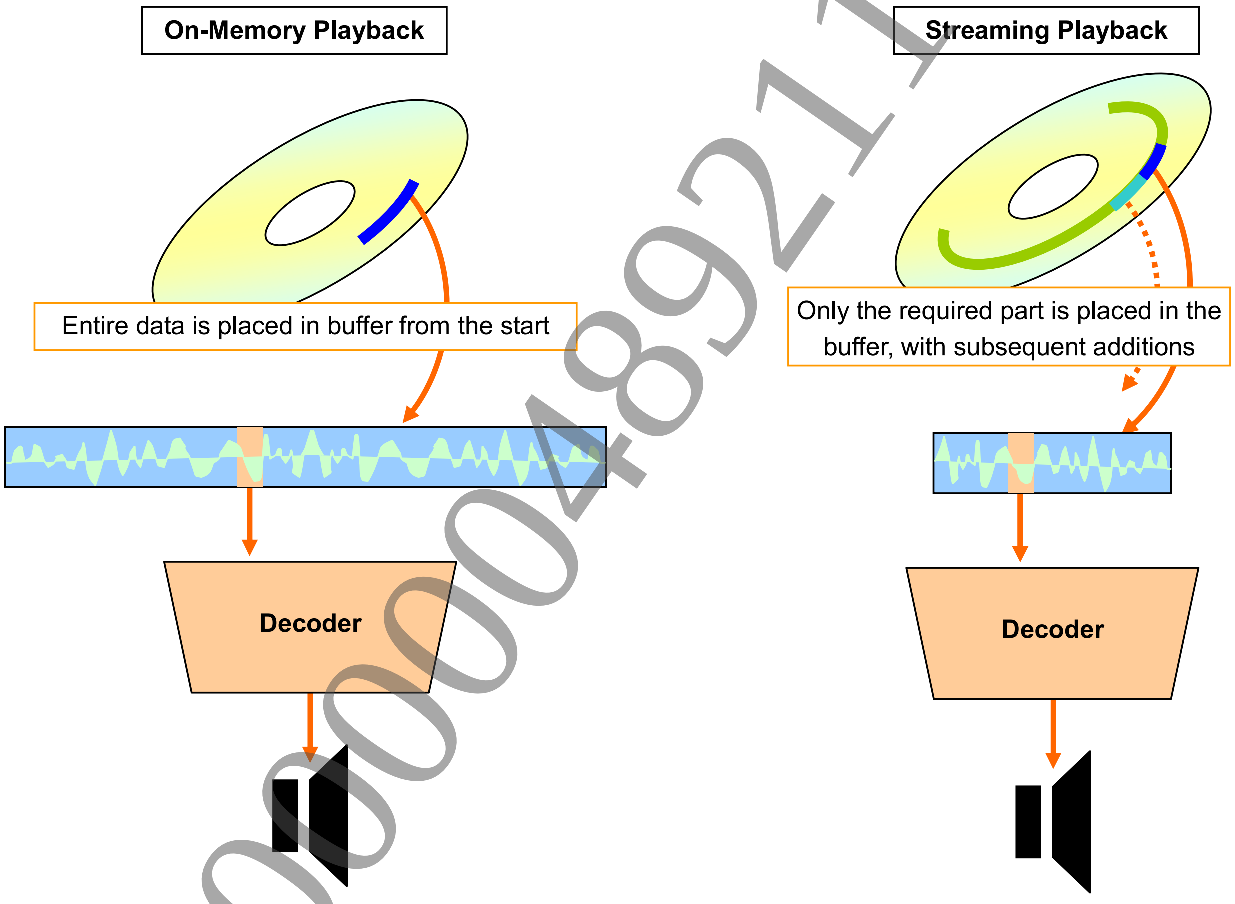
On-Memory Playback and Streaming Playback

Sound data can be played back either by on-memory processing or by streaming.

On-memory playback places the entire sound data on memory at once and plays it back. Although this requires a memory area the size of the data, this type of program is quite simple.

Streaming playback requires providing an input buffer of a certain size. It places only a part of the sound data on memory at a time for playback, and the succeeding data must be read into the input buffer as playback progresses. If there are delays in the reads of additional data, the sound will skip. This type of program is complicated because it requires measures to avoid such problems.

Figure 1 On-Memory Playback and Streaming Playback


On-Memory Playback
Streaming Playback
Entire data is placed in buffer from the start
Only the required part is placed in the buffer, with subsequent additions
Decoder
Decoder

Basic Procedure

libatrac serves to make streaming playback simple. Below, the procedure for streaming playback of data without a loop will be described. Except for the step of adding data, the procedure is the same for on-memory playback (of any data, with or without a loop).

(1) Inquiry of memory size required for creating a decoder group

Set *`size`*, *`wordLength`*, and *`totalCh`* of the `SceAtracDecoderGroup` structure, and call `sceAtracQueryDecoderGroupMemSize()`.

By calling this function, it is possible to obtain the memory size of the work memory required for creating a decoder group.

(2) Create a decoder group

After allocating the work memory required for creating a decoder group (a 256-byte alignment), call `sceAtracCreateDecoderGroup()`.

Calling this function creates a decoder group.

(3) Set the ATRAC9™ data and obtain the ATRAC™ handle

Provide a main buffer (a 256-byte alignment) for reading ATRAC9™ data, and load the ATRAC9™ data to that buffer. Next, set that buffer to an argument, and call `sceAtracSetDataAndAcquireHandle()`.

By calling this function, it is possible to obtain the ATRAC™ handle.

(4) Execute the decoding process

Allocate a buffer (a 256-byte alignment) for output. Setting the output buffer to an argument and calling `sceAtracDecode()` decode the ATRAC9™ data.

(5) Add ATRAC9™ data

The decoder status identifier, which can be obtained by calling `sceAtracDecode()`, can be used to determine whether ATRAC9™ data can be added to the buffer. When adding ATRAC9™ data to the buffer, the streaming information can be obtained with `sceAtracGetStreamInfo()`. For an application, use the streaming information to read the ATRAC9™ data. Next, use `sceAtracAddStreamData()` to send notification of the addition of stream data to the library.

(6) Free ATRAC™ handle

When ATRAC™ handle is no longer necessary, call `sceAtracReleaseHandle()` to free the ATRAC™ handle. This frees the main buffer and sub buffer associated with the ATRAC™ handle.

(7) Delete the decoder group

When the decoder group is no longer necessary, call `sceAtracDeleteDecoderGroup()` to delete the decoder group. This frees the work memory provided by calling `sceAtracCreateDecoderGroup()`.

Major APIs Used in Basic Processing

API	Description
`SceAtracDecoderGroup`	Decoder group structure
`SceAtracStreamInfo`	Streaming information structure
`sceAtracQueryDecoderGroupMemSize()`	Inquires memory size required for creating a decoder group
`sceAtracCreateDecoderGroup()`	Creates a decoder group
`sceAtracDeleteDecoderGroup()`	Deletes a decoder group

API	Description
`sceAtracSetDataAndAcquireHandle()`	Sets the ATRAC9™ data to be input and obtains ATRAC™ handle
`sceAtracReleaseHandle()`	Frees ATRAC™ handle
`sceAtracDecode()`	Executes the decoding process
`sceAtracGetStreamInfo()`	Obtains streaming information
`sceAtracAddStreamData()`	Sends notification of stream data addition

Changing the Playback Position

libatrac supports changes in the playback position during playback.

(1) Changes the Playback Position

Specify the playback position (in samples) to be changed to `sceAtracResetNextOutputPosition()` and call the function.

(2) Add ATRAC9™ data

If the playback position is changed during streaming playback, the ATRAC9™ data in the buffer is emptied, therefore, the ATRAC9™ data must be added. When adding ATRAC9™ data to the buffer, the streaming information can be obtained with `sceAtracGetStreamInfo()`. For an application, use the streaming information to read the ATRAC9™ data. Next, use `sceAtracAddStreamData()` to send notification of the addition of stream data to the library.

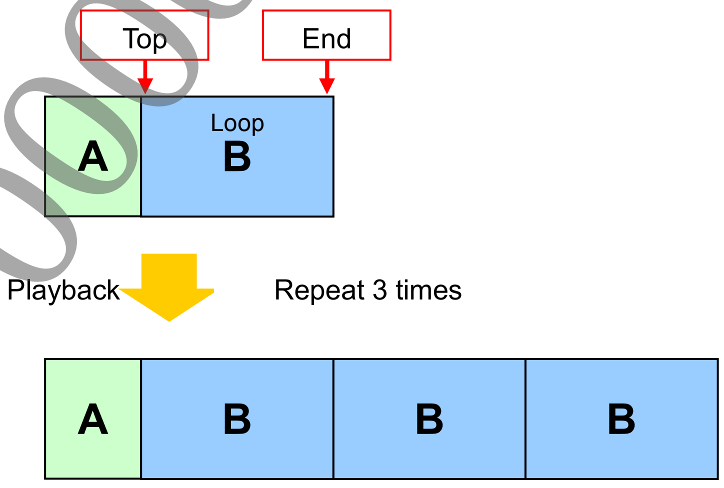
API for Changing the Playback Position

API	Description
`sceAtracResetNextOutputPosition()`	Resets the playback position

Loop Playback

libatrac supports playing back a certain specified section of sound data repeatedly. The loop (the section to repeat) can be specified when encoding the sound data, and the number of times to repeat playback of the loop can be specified at playback. Infinite repeat is supported, as well as changing the repeat number during playback.

Figure 2 Loop Playback



(1) Change the repeat number

Specify the number of loops to be changed to `sceAtracSetLoopNum()` and call the function.

(2) Loop playback

libatrac3plus manages loop playback, so playback processing by the application remains essentially the same as the basic procedure. Call `sceAtracDecode()` to perform decoding, and then use `sceAtracGetStreamInfo()` and `sceAtracAddStreamData()` to add data as necessary.

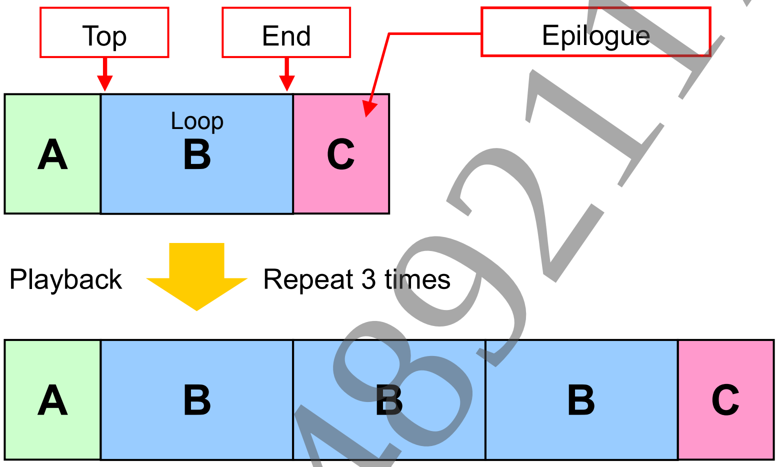
API for Loop Playback

API	Description
`sceAtracSetLoopNum()`	Sets number of times to repeat loop

Streaming Playback of Loop Data with an Epilogue

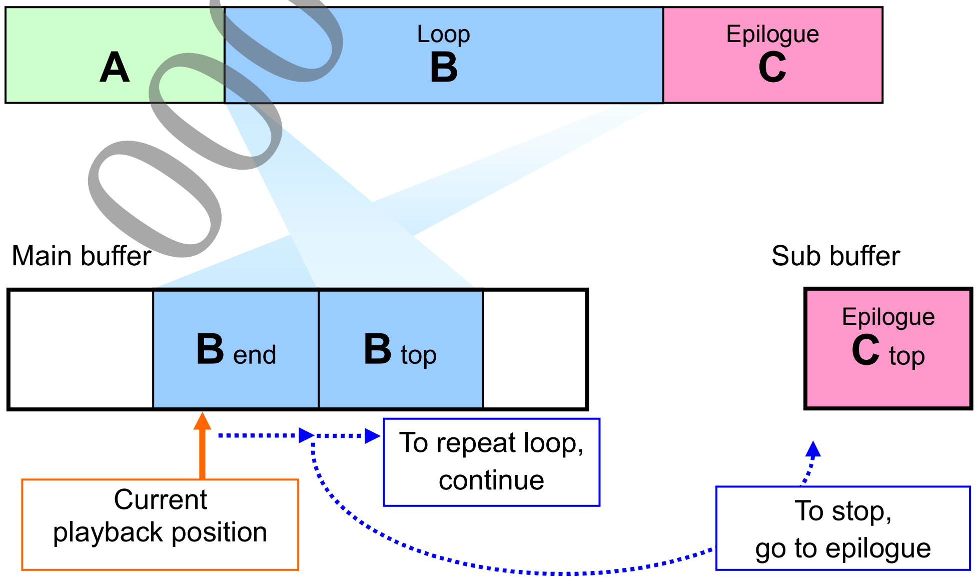
libatrac also supports a special loop playback in which an epilogue follows the loop section, as shown in the figure below.

Figure 3 Playback of Loop with Epilogue



When performing streaming playback for this data, a sub buffer may be required in addition to the normal input main buffer. A sub buffer provides on the memory both data that returns to the start of the loop and data that transitions to the epilogue when playing back data close to the end of the loop section.

Figure 4 Role of the Sub Buffer



(1) Check necessity of sub buffer

Call `sceAtracIsSubBufferNeeded()` to see if a sub buffer is necessary.

(2) Obtain sub buffer information

Call `sceAtracGetSubBufferInfo()` to obtain sub buffer information.

(3) Set sub buffer information

Provide a main buffer (a 256-byte alignment) for reading ATRAC9™ data, and load the ATRAC9™ data based on the sub buffer information. Next, set that buffer to an argument, and call `sceAtracSetSubBuffer()`.

Note that a large buffer size is not required as the sub buffer is only used temporarily so that data is not interrupted when transitioning to the epilogue section. However, if the reading of ATRAC9™ data does not catch up, the sound may be interrupted, so the buffer should have a sufficient size.

APIs for Streaming Playback of Data with an Epilogue

API	Description
`sceAtracIsSubBufferNeeded()`	Checks necessity of sub buffer
`sceAtracGetSubBufferInfo()`	Gets information of sub buffer
`sceAtracSetSubBuffer()`	Sets sub buffer

Procedure for Changing the Number of Output Samples

libatrac supports the feature for changing the number of output samples.

(1) Change the number of output samples

Specify the number of samples to be changed to `sceAtracSetOutputSamples()` and call the function.

Main API used in changing the number of output samples

API	Description
`sceAtracSetOutputSamples()`	Sets the number of output samples

Procedure for Obtaining the Decoder Group and Decoder Information

libatrac supports the features for obtaining the decoder group and decoder information.

Main APIs used for obtaining the decoder group and decoder information

API	Description
`SceAtracContentInfo`	Content information structure
`sceAtracGetDecoderGroupInfo()`	Gets decoder group information
`sceAtracGetContentInfo()`	Gets content information
`sceAtracGetLoopInfo()`	Gets loop information
`sceAtracGetOutputSamples()`	Gets the number of the output samples
`sceAtracGetNextOutputPosition()`	Gets output sample position
`sceAtracGetRemainSamples()`	Gets the number of remaining samples
`sceAtracGetOutputableSamples()`	Gets the number of outputable samples
`sceAtracGetDecoderStatus()`	Gets decoder state
`sceAtracGetVacantSize()`	Gets the free buffer size
`sceAtracGetInternalError()`	Gets Codec Engine internal error

3 Reference Information

ATRAC9™ Format

For ATRAC9™ file format, refer to the following document.

- ATRAC9™ File Format

4 Notes

Limitations

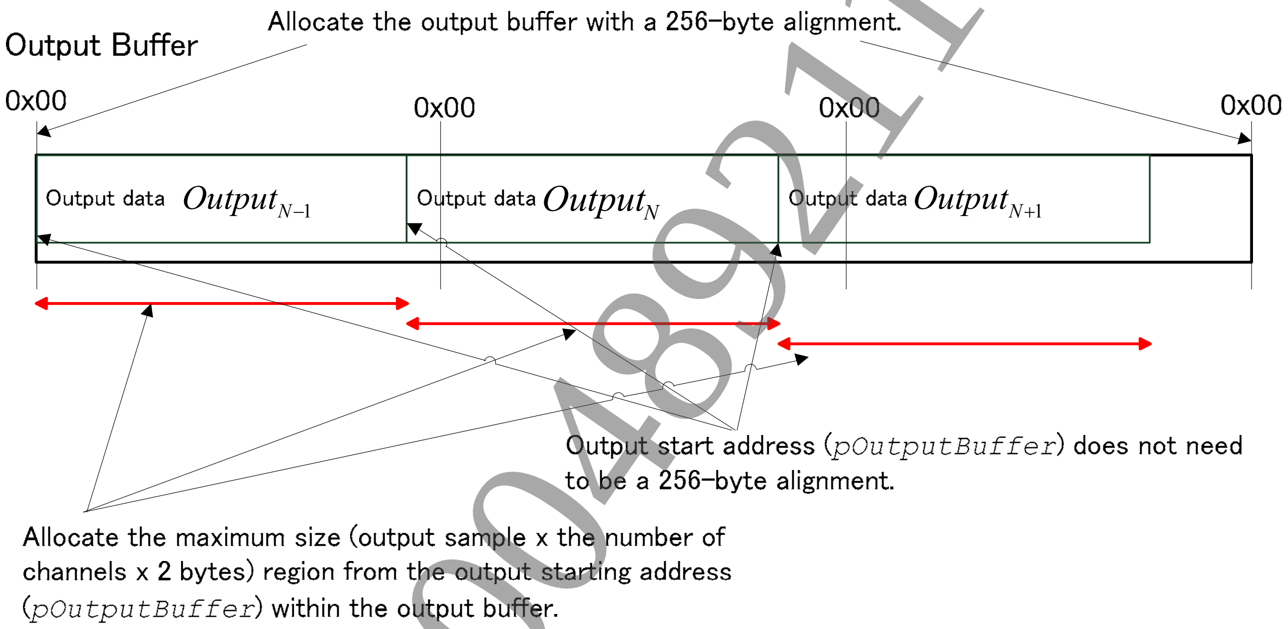
Allocating Memory Area

The work memory, main buffer memory, and sub buffer of libatrac must have a 256-byte alignment.

In addition, the output buffer has the following limitations. Also refer to Figure 5.

- Allocate the output buffer with a 256-byte alignment and a size of a multiple of 256-byte.
- The pointer *`pOutputBuffer`* of `sceAtracDecode()` must be allocated within the output buffer, but it does not have to have a 256-byte alignment.

Figure 5 Output Buffer


Output Buffer
Allocate the output buffer with a 256-byte alignment.
0x00
0x00
0x00
0x00
Output data $Output_{N-1}$
Output data $Output_{N}$
Output data $Output_{N+1}$
Output start address (*pOutputBuffer*) does not need to be a 256-byte alignment.
Allocate the maximum size (output sample x the number of channels x 2 bytes) region from the output starting address (*pOutputBuffer*) within the output buffer.

ATRAC9™ Band Extension Support

libatrac does not support the ATRAC9™ Band Extension, which is an extended format of ATRAC9™.

Check the value of `dwVersionInfo` in the RIFF Header to see whether the format is ATRAC9™ Band Extension. For details, refer to the "ATRAC9™ File Format" document.

Multiple Loops

libatrac does not have a feature for controlling multiple loops.

When data with multiple loops is set, only the start loop can be controlled.

Minimum Number of Loop Samples

libatrac has limitations on the number of loop samples.

Use content that has more than 3072 samples for the number of samples in the loop section.

Notes on Using a sub Buffer

When using a sub buffer in streaming playback, note that ATRAC9™ data reads and the decoding process must be synchronized. If the loop is being decoded when `sceAtracGetStreamInfo()` is called, this function will prioritize repeating the loop and request data of the loop. If the read is executed by an asynchronous thread, it is possible for loop data to be read and stored to the input buffer even if the decoding process now requires data of the epilogue.

There are two workarounds to this problem.

- When using a sub buffer, do not conduct asynchronous reads.
- To use a sub buffer and carry out asynchronous reads, call `sceAtracGetLoopInfo()` before and after the read to see if the loop state identifier has changed. If there is no change, call `sceAtracAddStreamData()` to notify addition of data. If there is a change, call `sceAtracGetStreamInfo()` again to receive the new instructions and read the necessary data.