

ATRAC9™ Simple Streaming Tutorial

© 2012 Sony Computer Entertainment Inc.
All Rights Reserved.
SCE Confidential

Table of Contents

About This Document	4
Purpose	4
Audience	4
Related Documentation	4
1 AT9 Simple Streaming Tutorial.....	5
Overview	5
NGS AT9 Streamer	5
FIOS Handler	5
NGS System Helper.....	5
Sulpha Common	5
NGS Common.....	5
Main Application and AT9 Streaming Class	5
Application	5
Interface	6
Usage.....	6
Configuration	6
Sample Configuration.....	7
Number of Voices.....	7
Audio Files to Stream.....	7
Audio Output Mode	7
Actions	7
Looping/Seeking Default Configuration.....	7
AT9 Streamer Configuration.....	8
Number of Streaming Buffers.....	8
Streaming Buffer Sizes	8
Looping Configuration.....	9
NGS Configuration	9
Number of Modules.....	9
System Granularity	9
Sample Rate	9
Voices Number of Channels	9
Number of Racks	9
Number of Voices.....	9
Default Volume.....	9
Maximum Pitch Ratio	9
FIOS Configuration	10
PS Archiver Usage.....	10
Sulpha Usage.....	10
Sulpha Enabling.....	10
Sulpha Mode.....	10
Sulpha Capture File Name.....	10
Razor Usage	10
2 AT9 Simple Streaming Source Code Overview	11
AT9 Simple Streaming Main Code	11
main()	11

init()	11
update().....	11
render()	11
shutdown()	11
handleInput().....	12
NGS AT9 Streamer.....	12
sceNgsAt9StreamerInit().....	12
sceNgsAt9StreamerSetLoop()	12
sceNgsAt9StreamerSeek()	12
sceNgsAt9StreamerPlay().....	12
sceNgsAt9StreamerStop()	12
sceNgsAt9StreamerRelease()	12
sceNgsAt9StreamerHandlePlayerCallback()	12
NGS System Helper	13
NGS Common	13
FIOS Handler	13
fiosHandlerInit().....	13
fiosHandlerTerminate().....	13
fiosHandlerOpen().....	13
fiosHandlerClose()	13
fiosHandlerReadSync().....	13
fiosHandlerReadAsync().....	13
Sulpha Common.....	13

About This Document

Purpose

This document provides an overview of the AT9 Simple Streaming functionality and implementation. The tutorial provides a sample implementation of AT9 streaming, using NGS as the sound synthesizer and FIOS2 to handle file access. The AT9 Simple Streaming Tutorial provides a reference implementation for use in application integration efforts.

Audience

This document is intended for application developers, whose main focus is PlayStation®Vita Audio development; aiming to implement a streaming solution to play their AT9 audio assets.

Related Documentation

Refer to the following related documents, included in the SDK, for further information on the following areas:

- NGS: See the *NGS Overview* and *NGS Reference*.
- FIOS: See the *libfios2 Overview* and *libfios2 Reference*.
- PS Archiver: See the *PSP2PSARC User's Guide* for usage information.
- Razor: See *Performance Analysis and GPU Debugging*.
- Sulpha: See the *libsulpha Overview* and *libsulpha Reference*.

1 AT9 Simple Streaming Tutorial

Overview

The AT9 Simple Streaming Tutorial shows how to implement an AT9 streaming solution, providing a simple interface to control streaming for playing, seeking, and looping.

The sample is structured using functional modules to facilitate integration in an independent manner.

NGS AT9 Streamer

This module allows you to instantiate audio streamers independently, and provides a straightforward interface for initializing the streamer to play, seek, and loop sections of an audio file.

The system requires that you initialize and set up the desired NGS configuration and FIOS. You may create multiple streamers to play specified audio files through NGS voices.

FIOS Handler

The File I/O Scheduler library can be used to schedule and manage I/O requests efficiently. The FIOS handler provides a wrapper to simplify initialization and usage of the library to access audio files. It uses the PS Archiver to read data from archives with multiple files. Note this may be preferable in order to transparently support access to multiple data types; however, there is no benefit to compressing already compressed data (such as AT9 files), and it is recommended that you use a specialized audio compressor.

NGS System Helper

The NGS System Helper is a module used to initialize and set up NGS, and to configure the audio graph and routings.

Sulpha Common

Sulpha Common is a module used to initialize and set up Sulpha, which is used for debugging. Sulpha allows capture and analysis of audio debug information at run time and can be used to easily track errors and for performance analysis.

NGS Common

The NGS Common is a group of helper functions used to initialize and set up the audio system. It contains methods for audio output configuration and debugging.

Main Application and AT9 Streaming Class

The main application is derived from the sample skeleton, and makes use of the common sample utilities to configure the sample graphics and display layout.

The AT9 Streaming class is the main application control class, where the system is set up and the logic to play, loop, and seek the audio files can be found. When the application is launched, all the modules required to run the sample are initialized, as is the AT9 streamer required to play the desired audio file. The user can then interact with the interface buttons to control the sample.

Application

The application provides a basic display with usage and debugging information.

The sample can be configured to play any AT9 files, and, by default, operates on the first file in the list.

Figure 1



Interface

- Δ: Show/Hide Debug Information
Debug information display control.
- X: Stream the whole file
Streams the audio file.
- O: Stream with looping enabled.
Configures the section to loop, either from the audio file header or using user-defined values (see [“Configuration”](#) for more information).
- □: Stream using seeking
Seeks and streams the section defined by the user (see [“Configuration”](#) for more information).

Usage

- (1) Build the sample in `\target\samples\sample_code\audio_video\tutorial_at9_simple_streaming`.
- (2) Launch `tutorial_at9_simple_streaming.self`.
- (3) Use the triangle, circle, square, and cross buttons to control the application.

Configuration

The sample is configurable so it can be customized to suit developer requirements. The following sections contain configuration details for each module:

- (1) AT9 Simple Streaming Sample
- (2) AT9 Streamers
- (3) NGS system
- (4) FIOS/PS Archiver

The sample also allows you to use Sulpha or Razor for debugging and performance analysis.

Sample Configuration

Number of Voices

The number of AT9 voices can be configured in "at9_simple_streaming.h". By default only one AT9 voice is used.

```
#define NGS_AT9_VOICES    (1)
```

Audio Files to Stream

The names of the files to be streamed are configured in the AT9Streaming class constructor (in "at9_simple_streaming.cpp").

```
m_pFiles[0] = MOUNTPOINT "/music.at9";
```

Note that the number of entries in m_pFiles is designed to match the number of voices (NGS_AT9_VOICES). However, the sample may be modified so the same streamer is reusable for streaming any file.

Audio Output Mode

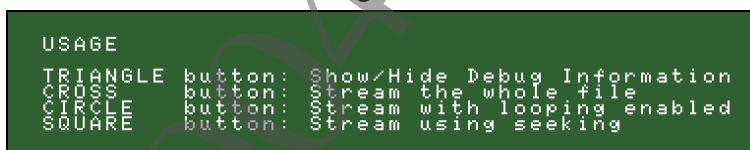
The sample can be configured in at9_simple_streaming.cpp to play the resulting audio or write the output to an audio file. Define OUTPUT_MODE as NGS_SEND_TO_DEVICE or NGS_WRITE_TO_FILE for those purposes. By default, the value of OUTPUT_MODE is NGS_SEND_TO_DEVICE.

```
#define OUTPUT_MODE (NGS_SEND_TO_DEVICE)
#define OUTPUT_FILE_NAME ("app0:out.raw")
```

Actions

The sample can be configured to perform the desired actions at run time by pressing the control buttons. See the "[Interface](#)" section for more details.

Figure 2



```
USAGE
TRIANGLE button: Show/Hide Debug Information
CROSS      button: Stream the whole file
CIRCLE     button: Stream with looping enabled
SQUARE     button: Stream using seeking
```

By default, the cross, circle and square buttons operate on the only AT9 file specified; however, the sample can be modified to suit the user's needs. You could, for example, have multiple files with each button operating on different files. If a voice is still playing as a result of a previous action when the user selects a new action, it is superseded by the new button action.

Looping/Seeking Default Configuration

The user can configure the sample offset (AT9_START_SAMPLE_OFFSET) and the number of samples to seek or loop (AT9_NUM_SAMPLES), making use of the definitions in at9_simple_streaming.cpp. The loop count is customized here.

```
#define AT9_START_SAMPLE_OFFSET    (72000)
#define AT9_NUM_SAMPLES            (96000)
#define AT9_LOOP_COUNT              (3)
```

AT9 Streamer Configuration

The AT9 streamers can be configured by modifying the definitions in `ngs_streamer_config.h`. Note that some restrictions apply to ensure efficient operation.

Number of Streaming Buffers

`NUM_STREAM_BUFFERS` is the number of streaming buffers used by each of the streamers. Default: 2.

```
#define NUM_STREAM_BUFFERS (2)
```

Streaming Buffer Sizes

`DEFAULT_AT9_STREAM_BUFFER_SIZE` is the size of each of the streaming buffers in KB. Default: 8 Kbytes.

```
#define DEFAULT_AT9_STREAM_BUFFER_SIZE (8 * 1024)
```

In order to configure this parameter to ensure the streamers' correct operation and performance, note the following:

(1) Minimum Buffer Size

In order to avoid data starvation in the system, the minimum recommended buffer size (in bytes) is:

```
((nGranularity * numChannels * MAX_PITCH_RATIO / AT9_SAMPLES_PER_PACKET)
 * packetEncodedSize) + packetEncodedSize
```

where:

- `nGranularity` – system granularity
- `numChannels` – number of audio channels in this voice
- `MAX_PITCH_RATIO` – the maximum value is 4 (output sample rate = 48kHz, input sample rate < 192kHz). It can be changed to a smaller value (for example, if you are only playing music at 48kHz, `MAX_PITCH_RATIO` can be set to 1).
- `AT9_SAMPLES_PER_PACKET` – samples per packet
- `packetEncodedSize` – AT9 packet encoded size (`nBlockAlign` in 'fmt' chunk)
- `(+ packetEncodedSize)` is added to ensure there is always enough data for the look ahead.

Note this is the recommended minimum size as it ensures every buffer contains enough data to be processed per update. It is possible to use a higher number of buffers of smaller size; however, only one callback is generated by the system per update, and it is the developer's responsibility to ensure all the processed buffers are filled with more data in time to avoid data starvation.

(2) Buffer Alignment

Ensure the streaming buffers are always aligned to the beginning of a packet or superpacket, depending on whether the superframe mode is OFF or ON. This simplifies the operation when using `nSamplesDiscardStart/nSamplesDiscardEnd`.

In order to avoid decode errors:

- When using `nSamplesDiscardStart`, ensure the buffer is always aligned to the beginning of a superpacket.
- When using `nSamplesDiscardEnd`, ensure the buffer is aligned to both the beginning and end of a superpacket.

Buffer Sizes vs. FIOS Performance

The buffer sizes affect the performance of the FIOS system. Using small buffers has a negative impact on FIOS performance, especially if the PS archiver is in use (the default in this sample).

Optimal buffer sizes balance memory usage with data access performance.

Looping Configuration

Use the `NGS_STREAMER_USE_HEADER_LOOPING_INFO` flag to determine whether the looping information is read from the AT9 file header or if the user prefers to manually configure it when calling `sceNgsAt9StreamerSetLoop()`. If the flag is passed in `nUseHeaderInfo` when calling this function, the loop start and loop end parameters are configured using the header information. The function defaults to the user-defined parameters if the AT9 header does not contain this information.

NGS Configuration

Configure NGS settings in `ngs_config.h`.

Number of Modules

`NUM_MODULES` defines how many unique module types NGS allows to be loaded. This value refers to the number of module types, regardless of the number of instances. It is set to 15 by default, as this is the current maximum number of available modules.

```
#define NUM_MODULES      (15)
```

System Granularity

`SYS_GRANULARITY` defines the PCM sample granularity. NGS processes and outputs PCM sample packets of this size, for every channel, with each update.

```
#define SYS_GRANULARITY  (512)
```

Sample Rate

`SYS_SAMPLE_RATE` defines the sample rate for NGS.

```
#define SYS_SAMPLE_RATE  (48000)
```

Voices Number of Channels

`NGS_VOICES_NUM_CHANNELS` is the default number of channels defined for the NGS voices.

```
#define NGS_VOICES_NUM_CHANNELS  (2)
```

Number of Racks

`NGS_NUM_RACKS` is the default number of racks in the system. By default, there are 2: the master rack and the AT9 player rack.

```
#define NGS_NUM_RACKS      (2)
```

Number of Voices

`NGS_NUM_VOICES` is the default number of voices in the system. By default there are 2: the master voice and the AT9 player voice.

```
#define NGS_NUM_VOICES     (2)
```

Default Volume

`NGS_DEFAULT_VOLUME` is the default volume set for the patches. The default is 1.0f.

```
#define NGS_DEFAULT_VOLUME  (1.0f)
```

Maximum Pitch Ratio

`MAX_PITCH_RATIO` is the maximum Pitch Ratio supported in the system. The default is 4 (output sample rate = 48kHz, input sample rate < 192kHz). It can be changed to a smaller value (for example, if you are only playing music at 48kHz, set `MAX_PITCH_RATIO` to 1).

```
#define MAX_PITCH_RATIO      (4)
```

FIOS Configuration

The FIOS module is initialized in the [fiosHandlerInit\(\)](#) function, where memory allocation and various other system parameters are configured. See the *libfios2 Overview* and *libfios2 Reference* for more information.

The FIOS handler module uses the PS archiver by default, but this feature can be switched off by disabling the flag `FIOS_PSARCHIVER_ENABLED` in `fios_handler.h`, which results in improved data reading performance, especially if archiver compression is in use.

```
#define FIOS_PSARCHIVER_ENABLED      (1)
```

PS Archiver Usage

The sample reads data from the `app0:/archive_data.psarc` archive file by default (if `FIOS_PSARCHIVER_ENABLED` is on), which is copied from the `/data` directory to `app0` when building the sample.

The archive included in the sample (in the `/data` folder) has been created with the PSP2PSARC tool with no compression. For more information on the format of the file and how to create your own, see *PSP2PSARC User's Guide*.

Sulpha Usage

To use Sulpha, enable the flags described below, either in the `sulpha_common.h` header file or as additional preprocessor definitions in the sample project.

The data captured can be interpreted by the Sulpha tool, which is included in the SDK. For more information, see the *libsulpha Overview* and *libsulpha Reference*.

Sulpha Enabling

The Sulpha functionality can be enabled with the `SULPHA_ENABLED` flag. The global Sulpha Capture enabling flag enables debugging data capture to be interpreted by the Sulpha tool. Output data is either sent to Sulpha in real time or to an output file, depending on the value of `SULPHA_LIVE_ENABLED`. The default is 0 (disabled).

Sulpha Mode

The `SULPHA_LIVE_ENABLED` flag enables or disables real-time capture in the Sulpha Tool. If it is enabled (`SULPHA_LIVE_ENABLED=1`), the debug information is visible in the Sulpha Tool at runtime. If it is disabled (`SULPHA_LIVE_ENABLED=0`), the debug data is saved in an output file for later use. The default is 0 (disabled).

Sulpha Capture File Name

If `SULPHA_LIVE_ENABLED` is 0, the module saves the Sulpha debug information to an output file defined as `CAPTURE_FILE_NAME`. By default, the file is saved to `app0:` and is named `out.sul`.

```
#define CAPTURE_FILE_NAME ("app0:out.sul")
```

Razor Usage

Razor enables performance data capture of the application at run-time. To enable Razor support in the sample, add `ENABLE_RAZOR_CAPTURE` to the list of preprocessor definitions in the sample project and rebuild. If the application is then loaded from the Razor plugin, the user will be able to capture performance data. Two markers have been configured by default: one to measure NGS system updates, and the other to analyze FIOS read timings.

2 AT9 Simple Streaming Source Code Overview

The AT9 Simple Streaming sample source code is split into the modules described in this section.

AT9 Simple Streaming Main Code

The application entry point is in `main.cpp`. The `AT9Streaming` class (derived from `SampleSkeleton`) contains the high-level application code (`at9_simple_streaming.h` and `at9_simple_streaming.cpp`). Key functions are described in the following sections.

main()

The application entry point. This function initializes the application using the [init\(\)](#) function, calls the [update\(\)](#) and [render\(\)](#) functions, which contain the main application code, and deallocates the system using [shutdown\(\)](#).

If Razor is enabled in the sample, the module is also loaded from here and used by the various modules in the sample.

init()

This function sets up the application by loading the necessary modules and initializing the system.

The first initializations begin in the base class (`SampleSkeleton::init()`), from which the `AT9Streaming` class derives, and the sample utilities used by the sample.

NGS is then initialized (`initNGS()`), as well as the audio graph, making use of the NGS system helper functions (including `createRack()` and `connectRacks()`), to create the desired components and routings. The FIOS handler initializer ([fiosHandlerInit\(\)](#)) is also invoked at this stage to set up the FIOS module for controlling file I/O.

The function then configures one streamer per audio file required by the sample, and prepares the audio output to either play the resulting audio or write it to an audio file.

Finally, the audio update and audio data threads are started. The audio update thread handles NGS updates and audio output. The audio data thread is used to handle NGS Player callbacks and read audio data into the relevant buffers.

update()

The main processing consists of an application loop that is terminated by pressing the PS button. The update function performs the following tasks:

- (1) Process buttons input to control the sample.
- (2) Wake up the Audio Data processing thread if there is any pending player callback to be processed.
- (3) Update the streamers' state in the display.

render()

Calls in the main processing loop responsible for the graphics rendering.

shutdown()

Shuts down the application by releasing the resources allocated in the [init\(\)](#) function and unloading the relevant system modules. `SampleSkeleton::shutdown()` deallocates the base class components and must be called at the end of this function.

handleInput()

Provides the sample's input interface handling and provides methods to play, seek, and loop streamed audio.

NGS AT9 Streamer

The AT9 Streamer module is defined in `ngs_at9_streamer.h` and `ngs_at9_streamer.c`. The module provides a simple interface to stream an AT9 file, using the name of the file and the NGS voice handle to stream it. It also provides an interface to set up the streamer to loop and seek within the audio file.

The main functions are described below:

sceNgsAt9StreamerInit()

AT9 Streamer initialization function. It initializes the voice and resources needed.

sceNgsAt9StreamerSetLoop()

AT9 Streamer function to set up an audio loop and initialize the streamer to play the different sections in the file a specified number of times. Only one loop is supported at any given time, and calling this function multiple times will overwrite the looping information.

Looping can be configured based either on the parameters (`nLoopStart` and `nLoopEnd`) if `nUseHeaderInfo` is 0, or the file's header looping information if the flag `NGS_STREAMER_USE_HEADER_LOOPING_INFO` is passed in `nUseHeaderInfo`.

sceNgsAt9StreamerSeek()

AT9 Streamer function to seek a specified section to play when the user calls [sceNgsAt9StreamerPlay\(\)](#). The function receives an offset and the number of samples, and configures the streamer to play only this section.

sceNgsAt9StreamerPlay()

AT9 Streamer playing start function. It fills the streaming buffers with data and sets up the voice to start playing. Note that the user must configure the streamer to play the desired section, even if it is the complete file, by calling [sceNgsAt9StreamerSeek\(\)](#) or [sceNgsAt9StreamerSetLoop\(\)](#), prior to playing.

sceNgsAt9StreamerStop()

AT9 Streamer playing stop function. It stops the playback and disables the voice.

sceNgsAt9StreamerRelease()

AT9 Streamer release function. It releases the resources allocated in [sceNgsAt9StreamerInit\(\)](#).

sceNgsAt9StreamerHandlePlayerCallback()

AT9 Streamer function called to handle the NGS player callbacks. The function updates the streamer status and fills in the streaming buffers with more data if necessary.

This function is provided to allow the user to move the data reading to a separate thread and avoid blocking the callback generator thread. The data reading can be slow depending on the configuration, so this allows for improved performance optimization.

NGS System Helper

The group of helper functions in `ngs_system_helper.h` and `ngs_system_helper.c` are designed to facilitate the NGS system initialization and setup. It provides functions to initialize NGS (`initNGS()`), create and connect racks (`createRack()` and `connectRacks()`) and set volumes in a specified patch (`setPatchVolume()`).

NGS Common

The helper functions in `ngs_common.h` and `ngs_common.c` include audio output handling functionality (`prepareAudioOut()`, `writeAudioOut()` and `shutdownAudioOut()`) and other debugging utilities, such as Razor performance counter initialization (`threadPerInit()`) and NGS debug information (`printParamError()`).

FIOS Handler

The FIOS handling module is defined in `fios_handler.h` and `fios_handler.c`. The module provides a wrapper around the FIOS library, with a simple interface to initialize and control the system.

The main functions are described below:

fiosHandlerInit()

FIOS initialization function. It initializes the FIOS system and resources needed for its correct operation.

fiosHandlerTerminate()

FIOS deallocation function. It deallocates the FIOS system and resources allocated in [fiosHandlerInit\(\)](#).

fiosHandlerOpen()

This function opens a file in the FIOS system. It receives the name of the file to be opened and returns a file handle.

fiosHandlerClose()

This function closes a file in the FIOS system, using the associated file handle.

fiosHandlerReadSync()

This function reads synchronously from the specified file.

By default, the FIOS system is configured to read at the maximum priority and to deliver the data at the earliest time. However, this can be modified to suit user requirements for I/O access.

fiosHandlerReadAsync()

This function starts an asynchronous read from the specified file. Note that the data will not be available in the relevant buffer until the callback `fiosHandlerReadCallback()` is received.

By default, the FIOS system is configured to read at the maximum priority and to deliver the data at the earliest time. However, this can be modified to suit user requirements for I/O access.

Sulpha Common

The Sulpha handling functions in `sulpha_common.h` and `sulpha_common.c` provide an interface to initialize (`sulphaTracingStart()`) and deallocate (`sulphaTracingStop()`) Sulpha, includes an update function (`sulphaTracingUpdate()`) to regularly update the system, and provides message tracing functionality (`sulphaTracingMessage()`) to facilitate debugging.