

Streaming Controller 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 Streaming Controller Tutorial.....	5
Overview	5
NGS Streamer.....	5
NGS AT9 Streamer	5
NGS PCM Streamer.....	5
FIOS Handler	5
NGS System Helper.....	5
Sulpha Common	5
NGS Common.....	6
Main Application and Streaming Controller Class.....	6
Application	6
Interface	6
Usage.....	6
Configuration	7
Sample Configuration.....	7
Number of Voices.....	7
Audio Files to Stream.....	7
Audio Output Mode	7
Actions	7
Cross-fading Default Configuration.....	8
Streamers Configuration	8
Number of Streaming Buffers.....	8
Streaming Buffer Sizes	8
AT9 Streaming Buffer Sizes	8
PCM/ADPCM Streaming Buffer Sizes	9
Looping Configuration.....	9
NGS Configuration	9
Number of Modules.....	9
System Granularity	9
Sample Rate	9
Voices Number of Channels	10
Number of Racks	10
Number of Voices.....	10
Default Volume.....	10
Maximum Pitch Ratio	10
FIOS Configuration	10
PS Archiver Usage	10
Sulpha Usage.....	10
Sulpha Enabling.....	11
Sulpha Mode.....	11
Sulpha Capture File Name.....	11
Razor Usage	11

2 Streaming Controller Source Code Overview.....	12
Streaming Controller Main Code	12
main()	12
init()	12
update()	12
render()	12
shutdown()	12
handleInput()	13
NGS Streamer	13
sceNgsStreamerInit()	13
sceNgsStreamerSetLoop()	13
sceNgsStreamerSeek()	13
sceNgsStreamerPlay()	13
sceNgsStreamerStop()	13
sceNgsStreamerRelease()	13
sceNgsStreamerHandlePlayerCallback()	13
AT9 and PCM NGS Streamers	14
NGS System Helper	14
NGS Common	14
FIOS Handler	14
fiosHandlerInit()	14
fiosHandlerTerminate()	14
fiosHandlerOpen()	14
fiosHandlerClose()	14
fiosHandlerReadSync()	14
fiosHandlerReadAsync()	14
Sulpha Common	15

About This Document

Purpose

This document provides an overview of the Streaming Controller functionality and implementation. The tutorial provides a sample implementation of streaming for different types of audio files, using NGS as the sound synthesizer and FIOS2 to handle file access. This 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 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 Streaming Controller Tutorial

Overview

The Streaming Controller Tutorial shows how to implement an audio streaming solution, providing a simple interface to control streaming for any of the following formats: AT9, VAG or PCM. The sample also provides cross-fading functionality to transition smoothly between different audio streams and play a series of streams in a chain.

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

NGS Streamer

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

The NGS Streamer uses the NGS AT9 Streamer and NGS PCM Streamer modules to handle the characteristics of the different audio formats. Any of these modules can be used independently. The NGS Streamer operates as an abstraction layer: the user can stream any type of supported audio file and it will be configured automatically.

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.

NGS AT9 Streamer

This module allows you to instantiate AT9 streamers independently and provides a straightforward interface for initializing the streamer to play, seek, and loop sections of an AT9 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.

NGS PCM Streamer

This module allows you to instantiate PCM streamers independently, and provides a straightforward interface for initializing the streamer to play, seek, and loop sections of a PCM or ADPCM 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 Streaming Controller Class

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

The Streaming Controller class is the main application control class, where the system is set up and the logic to play the audio files, cross-fade between them, and perform chain handling can be found. When the application is launched, all the modules to run the sample are initialized, as are the streamers to play the desired audio files. 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, VAG or PCM file files, and, by default, operates on the first two files in the list.

Figure 1



```

STREAMING CONTROLLER SAMPLE
File0: /archive/mount/point/music.at9
File1: /archive/mount/point/stereo_str-LR.vag

USAGE
TRIANGLE button: Show/Hide Debug Information
CIRCLE   button: Play file0 with cross-fading
CROSS    button: Play file1 with cross-fading
SQUARE   button: Stream all files as a chain

STREAMER0 CURRENTLY PLAYING
STREAMER1 CURRENTLY PLAYING

```

Interface

- Δ: Show/Hide Debug Information.
Debug information display control.
- O: Play file0 with cross-fading.
Streams the first audio file in the list (file0). If file1 is playing, it cross-fades between the two streams.
- X: Play file1 with cross-fading.
Streams the first audio file in the list (file1). If file0 is playing, it cross-fades between the two streams.
- □: Stream all files as a chain.
Streams all the files in the list as a chain.

Usage

- (1) Build the sample in
`\target\samples\sample_code\audio_video\tutorial_streaming_controller.`
- (2) Launch `tutorial_streaming_controller.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) Streaming Controller Sample
- (2) 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 voices can be configured in `streaming_controller.h`. By default an AT9 voice and ADPCM voice are used.

```
#define NGS_AT9_VOICES           (1)
#define NGS_PCM_VOICES          (0)
#define NGS_ADPCM_VOICES        (1)
```

Audio Files to Stream

The names of the files to be streamed are configured in the `StreamingController` class constructor in `streaming_controller.cpp`.

```
m_pFiles[0] = MOUNTPOINT "/music.at9";
m_pFiles[1] = MOUNTPOINT "/stereo_str-LR.vag";
```

Note that the number of entries in `m_pFiles` is designed to match the number of voices (`NGS_SOURCE_VOICES`). However, the sample may be modified so the same streamer is reusable for streaming multiple files of the same type (the PCM streamer can be used for both PCM and ADPCM).

Audio Output Mode

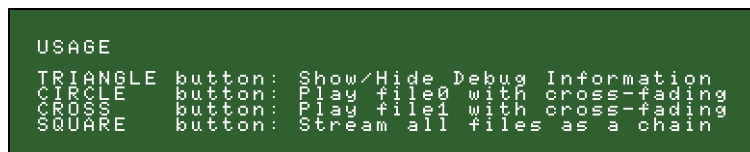
The sample can be configured to play the resulting audio or write the output to an audio file in `streaming_controller.cpp`. Define `OUTPUT_MODE` as `NGS_SEND_TO_DEVICE` or `NGS_WRITE_TO_FILE` for those purposes. By default, the value of `OUTPUT_MODE` is `NGS_SEND_TO_DEVICE`.

```
#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
CIRCLE   button: Play file0 with cross-fading
CROSS    button: Play file1 with cross-fading
SQUARE   button: Stream all files as a chain
```

By default, the cross, circle and square buttons operate on the file specified; however, the sample can be modified to suit the user's needs. 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.

Cross-fading Default Configuration

The user can configure the cross-fading period using `NGS_STREAMER_CROSSFADING_UPDATES` in `streaming_controller.cpp`. With each update, the system modifies the streamers' volumes for the specified number of updates.

```
#define NGS_STREAMER_CROSSFADING_UPDATES (200)
```

The sample implements a sinusoidal cross-fading and the `kCosine` definition in the same file, providing values of the cosine function for the range $(0, \pi/2)$ to optimize runtime calculations. If the user modifies the `NGS_STREAMER_CROSSFADING_UPDATES` value, the definition of `kCosine` must be updated to match the correct number of entries.

Streamers Configuration

The 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_STREAM_BUFFER_SIZE` and `DEFAULT_AT9_STREAM_BUFFER_SIZE` are the sizes (in KB) of the PCM and AT9 streaming buffers, respectively. Default: 8 Kbytes.

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

It is necessary to configure this parameter to ensure the streamers' correct operation and performance. Some of the restrictions, such as minimum buffer size and buffer alignment, are audio type specific, and are described in separate sections.

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.

AT9 Streaming Buffer Sizes

(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:

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

PCM/ADPCM Streaming Buffer Sizes

(1) Minimum Buffer Size

For PCM and ADPCM formats, ensure there is enough data in the buffers to avoid data starvation in the system at all times.

(2) Buffer Alignment

Align ADPCM buffers to the encoded block size (`VAG_FILE_DATA_BLOCK_SIZE`) to ensure that full blocks are always available for decoding.

Looping Configuration

Use the `NGS_STREAMER_USE_HEADER_LOOPING_INFO` flag to determine whether the looping information is read from the file header or if the user prefers to manually configure it when calling `sceNgsStreamerSetLoop()`. This is only supported for AT9 and PCM types. 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 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 3 racks, the master rack, the PCM player rack and the AT9 player rack.

```
#define NGS_NUM_RACKS                (3)
```

Number of Voices

NGS_NUM_VOICES is the default number of voices in the system. By default there are 3 voices, the master voice, the PCM player voice and the AT9 player voice.

```
#define NGS_NUM_VOICES                (3)
```

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 (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 the *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 Streaming Controller Source Code Overview

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

Streaming Controller Main Code

The application entry point is in `main.cpp`. The `StreamingController` class (derived from `SampleSkeleton`) contains the high-level application code (`streaming_controller.h` and `streaming_controller.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 `StreamingController` 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 the streamers as audio files 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) Wakes up the Audio Data processing thread if there is any pending player callback to be processed.
- (2) Updates the volumes on the NGS patches if a cross-fading transition between two streamers is ongoing
- (3) Process buttons input to control the sample.
- (4) Updates 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 the streamers available in the sample, cross-fading between them to provide smooth transitions, and chaining all the streamers to play the entire list of audio files.

NGS Streamer

The Streamer module is defined in `ngs_streamer.h` and `ngs_streamer.c`. The module provides a simple interface to stream any supported audio file (AT9, PCM, or VAG), 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:

sceNgsStreamerInit()

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

sceNgsStreamerSetLoop()

Streamer function to setup 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 passed in (`nLoopStart` and `nLoopEnd`) if `nUseHeaderInfo` is 0, or the file's header looping information (only supported for AT9 and PCM) if the flag `NGS_STREAMER_USE_HEADER_LOOPING_INFO` is passed in `nUseHeaderInfo`.

sceNgsStreamerSeek()

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

sceNgsStreamerPlay()

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 [sceNgsStreamerSeek\(\)](#) or [sceNgsStreamerSetLoop\(\)](#) prior to playing.

sceNgsStreamerStop()

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

sceNgsStreamerRelease()

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

sceNgsStreamerHandlePlayerCallback()

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.

AT9 and PCM NGS Streamers

The NGS Streamer module uses the AT9 NGS Streamer (see `ngs_at9_streamer.h` and `ngs_at9_streamer.c`) and the PCM NGS Streamer (see `ngs_pcm_streamer.h` and `ngs_pcm_streamer.c`) to handle the operation of AT9 streaming and PCM/ADPCM streaming, respectively. The interface of these two modules is similar to the NGS Streamer itself, as this is just a wrapper to provide transparent handling of any audio type.

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's requirements for the different I/O accesses.

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.

000004892117