

N I N T E N D O
NITRO-System
NITRO-Composer
Sound System Manual

Version 1.4.1a

The contents in this document are highly
confidential and should be handled accordingly.

Confidential

These coded instructions, statements, and computer programs contain proprietary information of Nintendo of America Inc. and/or Nintendo Company Ltd. and are protected by Federal copyright law. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

Table of Contents

1	Introduction	7
2	Hardware.....	8
2.1	Sound Circuit.....	8
2.2	Sound.....	8
2.2.1	ADPCM/PCM	8
2.2.2	PSG Rectangular Wave	9
2.2.3	Noise.....	9
2.3	Channel Numbers.....	9
2.4	Capture.....	9
3	Player.....	10
3.1	What is a Player?	10
3.2	The Number of Players.....	10
3.3	Limitations on Simultaneous Sequence Playing.....	10
3.3.1	System Limitation.....	10
3.3.2	Player Limitation.....	11
3.4	NITRO-Composer Tracks	11
3.4.1	The Number of Tracks.....	11
3.5	Configuring Channel Limitations for Sequence Playback to Avoid Channel Conflict	11
3.5.1	How to Configure the Channel Limitations	12
3.5.2	Channel Limitations on Each Track.....	12
3.6	Differences With the Player in the Sound Driver.....	12
3.6.1	Cannot Use Concurrently with the Player in the Sound Driver	12
4	Priority.....	13
4.1	What is Priority?	13
4.2	Player Priority	13
4.3	Voice Priority	14
4.3.1	What is Voice Priority?	14
4.3.2	Setting Up Voice Priority.....	14
4.3.3	Release	15
5	Memory Management	16
5.1	The Need for Memory Management.....	16
5.2	Two Types of Heap.....	16

5.3	The Sound Heap	17
5.3.1	About the Sound Heap	17
5.3.2	How to Use the Sound Heap	17
5.3.3	Group Load	17
5.4	The Player Heap.....	18
5.4.1	About the Player Heap	18
5.4.1.1	Cautions for Sequence Archive Playback	18
5.4.2	How to Use the Player Heap	18
5.4.3	Creating the Player Heap	18
5.5	A Comparison of the Sound Heap and the Player Heap.....	19
5.6	An Example of Memory Management	20
5.6.1	Specifications	20
5.6.2	Preparation.....	20
5.6.2.1	Creating the Group	20
5.6.2.2	Configuring the Player Heap.....	20
5.6.2.3	Confirm the Configuration.....	21
5.6.3	Startup Processing	21
5.6.3.1	Creating the Sound Heap	21
5.6.3.2	Initializing the Sound Archive.....	21
5.6.3.3	Creating the Player Heap	21
5.6.3.4	Loading Resident Groups	21
5.6.4	Gameplay Processing	21
5.6.4.1	Loading to the Swap Region.....	21
5.6.4.2	Background Music Playback.....	21
5.6.4.3	Switching Scenes	21
5.6.5	Memory Usage Conditions	22
6	Stream Playback	23
6.1	What is Stream Playback?.....	23
6.2	The Mechanism of Stream Playback	23
6.2.1	Allocate Channels	23
6.2.2	Prepare the Data	23
6.2.3	Start Playback	23
6.2.4	Load Data as Needed.....	24
6.3	Differences Between Stream and Sequence Playback.....	24

7	Output Effects	25
7.1	What are Output Effects?	25
7.2	Precautions About Using Output Effects.....	25
7.2.1	Reduced Number of Simultaneous Sounds Generated.....	25
7.2.2	Processing Load	25
7.2.3	Sound Distortion.....	25
7.3	Precautions when Using Surround	26
7.3.1	Increase in Volume.....	26
7.3.2	Difference in Balance Between the Left and Right Components	26
7.4	About Stereo and Mono Output	26
7.4.1	When Using All Output Effects	26
7.4.2	When Using Only Surround and Headphone Output Effects.....	26
8	Sound Processing.....	27
8.1	ARM7	27
8.2	Main Memory Access	27
8.2.1	Waveform Data Transfer	27
8.2.2	Sequence Data Processing.....	27
8.3	Sound Processing Units	27
8.3.1	Quarter Note Resolution.....	27

Tables

Table 2-1	The Functionality of Each Channel Number	9
Table 5-1	A Comparison of the Characteristics and Primary Uses of the Sound and Player Heap	19
Table 6-1	Characteristics of Sequence and Stream Playback.....	24
Table 7-1	Output Effect Types	25

Figures

Figure 2-1	Sound Circuit	8
Figure 3-1	Conceptual Diagram of Player	10
Figure 3-2	Conceptual Diagram of Tracks.....	11
Figure 4-1	Example of Player Priority.....	14
Figure 5-1	Sound Heap Allocation and Deallocation	17
Figure 5-2	The Player Heap Instructions.....	18
Figure 5-3	An Example of Memory Management	20
Figure 5-4	Memory Usage Conditions.....	22

Revision History

Version	Revision Date	Description
1.4.1a	2007/04/27	Corrected typographical errors. Changed dates in Revision History to international format.
1.4.1	2005/09/01	Corrected typographical errors.
1.4.0	2005/03/28	Revised the player heap description because the bank and waveform data necessary for sequence playback are enabled to be loaded to the player heap. Added a description of the sound driver player. Revised the description of how to check the configuration as a practical example of memory management. Added descriptions of the normal and mono output effect.
1.3.0	2005/01/31	Added a description related to the limitations on the sequence channels. Changed "NITRO" to "Nintendo DS".
1.2.0	2004/12/06	Added description of output effects.
1.1.0	2004/08/10	Added description of channel numbers. Added description of stream playback. Changed "sound" to "channel" in sound circuit section.
1.0.0	2004/07/20	Style adjustments.
0.2.0	2004/06/01	Added a chapter on memory management. Made revisions involving the multiple sequences becoming able to be played on one player. Made revisions involving the changes in the handling of the player priority. Changed "Maximum Simultaneous Number of Sequences Playing" to "Maximum Number of Simultaneous Sequences Playing".
0.1.0	2004/04/01	Initial Version.

1 Introduction

This document provides a basic description of NITRO-Composer.

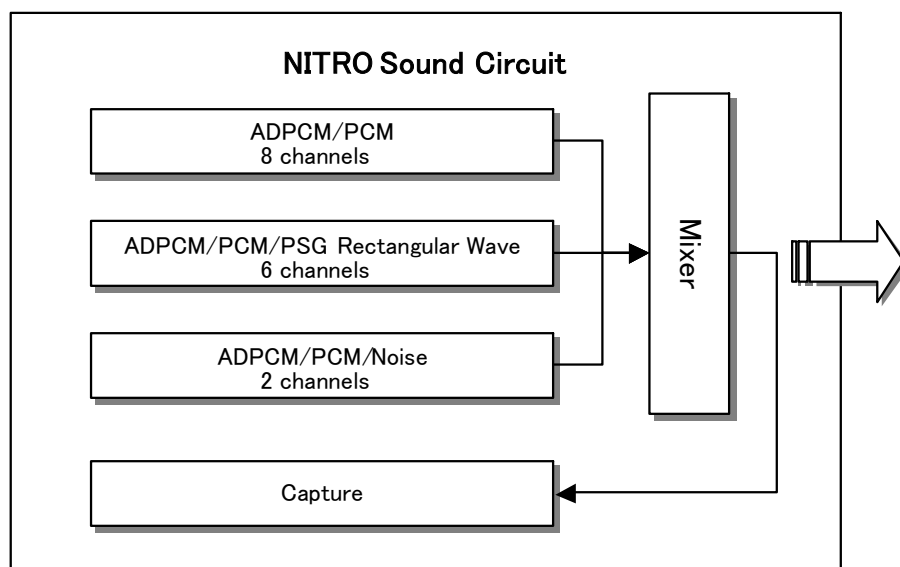
2 Hardware

This chapter describes the sound circuit for the Nintendo DS (DS).

2.1 Sound Circuit

The sound circuit of the DS is illustrated in Figure 2-1.

Figure 2-1 Sound Circuit



2.2 Sound

Figure 2-1 represents the sound circuit that comprises the channels for generating sounds.

All 16 channels are configured for ADPCM/PCM playback. Eight channels are reserved exclusively for ADPCM/PCM playback only, six channels can also play PSG rectangular waves, and two channels can also play noise. Each channel creates a single sound and up to 16 sounds can be played simultaneously.

However, the PSG rectangular wave and noise channels are limited. Therefore, it is not possible to simultaneously play 16 sounds only with PSG rectangular waves or noise.

2.2.1 ADPCM/PCM

ADPCM or PCM data can be played. PCM can be played back with 8 bits or 16 bits.

2.2.2 PSG Rectangular Wave

PSG rectangular waves can be played. The duty ratio can be selected.

2.2.3 Noise

White noise can be played.

2.3 Channel Numbers

The 16 channels are numbered from 0 to 15. Each channel has a slightly different function outlined in Table 2-1.

Table 2-1 The Functionality of Each Channel Number

Channel Number	Function
0, 2	Plays ADPCM/PCM. The output of these channels can be used for sound capture input.
1, 3	Plays ADPCM/PCM. Because these channels are used for both sound capture and the timer, when using sound capture, they can be used only as output channels.
4 through 7	Plays ADPCM/PCM.
8 through 13	Plays ADPCM/PCM and PSG rectangular wave.
14, 15	Plays ADPCM/PCM and white noise.

2.4 Capture

The capture component in the sound circuit loads and writes the data in memory. In Figure 2-1, the capture component can take the mixer, channel 0, or channel 2 output as input.

Using a feedback loop between the mixer and capture component, the output is resent to the mixer which creates a time delay. This time delay creates a reverb effect in the sound.

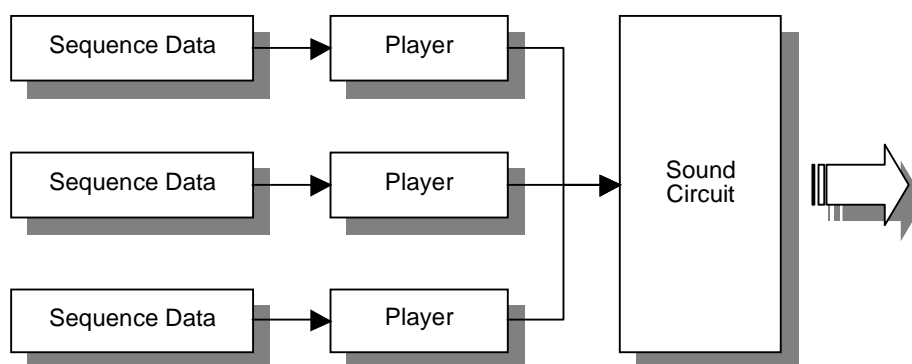
Using sound capture, NITRO-Composer provides functions for sound effects such as `surround`. For details, see Chapter 7 Output Effect.

3 Player

3.1 What is a Player?

When using NITRO-Composer, the channels in the sound circuit are not usually operated directly. Sound is played as sequence data units using a player.

Figure 3-1 Conceptual Diagram of Player



By default, a player can play only one sequence data unit, but a player can be configured to play multiple sequence data units.

3.2 The Number of Players

There are 32 players. A number from 0 to 31 is assigned to each player. These assigned numbers are known as player numbers. When playing back sequence data, the player that will play the sequence must be specified by its player number.

3.3 Limitations on Simultaneous Sequence Playing

Two factors limit the maximum number of sequences that can be played simultaneously. The first limitation is the total number of sequences that can be played on the system. The second limitation is the number of sequences that can be played on each player.

3.3.1 System Limitation

The system can play a maximum of sixteen sequences simultaneously. There is no distinction between BGM and soundtrack playback. For example, up to fifteen soundtracks can be played simultaneously with one a BGM sequence.

3.3.2 Player Limitation

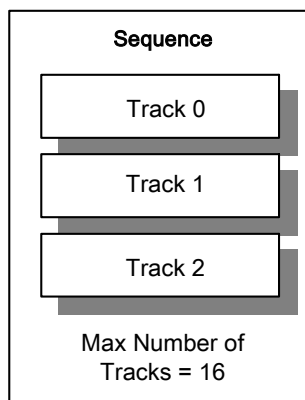
The number of sequences that can be played on one player is limited. By default, one sequence can be played back on each player. However, this limitation can be specified by setting the player number to 16, there is no limit on the number of sequences that a player can play.

Unlike the system limitation, the player limitation is set intentionally. For example, if one player is configured to play the dialogue for the main character and the sequence limit is one, an error will result when two lines of dialogue play simultaneously. This feature is useful when you want to play multiple sound effects and dialog for more than one character

3.4 NITRO-Composer Tracks

In MIDI sequence data there is a concept known as a track. In a single sequence there are multiple tracks. Playing separate sequences for each track makes up one song. NITRO-Composer also uses tracks.

Figure 3-2 Conceptual Diagram of Tracks



Note that there are commands and related functions for sequences as well as for individual tracks.

3.4.1 The Number of Tracks

Up to a maximum of sixteen tracks can be used for one sequence. However, there are only 32 tracks overall on the system. To play one sequence, at least one track must be defined.

For example, if two sequences each use sixteen tracks, the requirement to play a sequence is met. But because all the tracks are used, no additional sequences can be played.

3.5 Configuring Channel Limitations for Sequence Playback to Avoid Channel Conflict

To play a sequence, all the channels are usually searched to find usable channels. However, sequence playback is limited to certain channels on different players.

For example, channel limitations may need to be set to avoid channel conflict under the following conditions:

- If sound capture starts in the middle of sequence playback, the generated sounds on channels 1 and 3 are stopped because channels 1 and 3 can be used for sound capture. If you want this sequence to keep playing, do not use channels 1 and 3.
- To ensure that the sound effects and background music do not cancel each other out, apply a restriction that the channels used by the sound effects and the background music player do not overlap.

3.5.1 How to Configure the Channel Limitations

The sound designer configures channel limitations on each player. If the channel limitation of a player is not configured, the player is treated as having no channel limitations.

The channel limitation configured for each player by the sound designer can be overwritten by the programmer at execution.

3.5.2 Channel Limitations on Each Track

The channel limitations configured for each player effect all sequence playback on that player. The programmer can configure the player at execution so that limitations only apply to particular tracks on a given sequence.

These changes made by the programmer are independent of the player and are valid only for the designated tracks of a designated sequence. The configurations for each track become valid when these settings combine with the channel limitations for each player.

3.6 Differences With the Player in the Sound Driver

There is a player in the sound driver, but the player in NITRO-Composer has been expanded so that it is easier to use than the player in the sound driver.

For example, while the player in the sound driver can only play one sequence per player, the NITRO-Composer player allows playback of multiple sequences.

3.6.1 Cannot Use Concurrently with the Player in the Sound Driver

The NITRO-Composer player cannot be used concurrently with the sound driver player.

In addition, the player number for NITRO-Composer and the Sound Driver are different. If a certain player number is not used by NITRO-Composer, it does not mean that the player number can be used by the sound driver.

4 Priority

4.1 What is Priority?

As described in Chapter 3, the number of sequences that can be played concurrently and the number of channels are limited. However, remaining within the limits and creating data or carrying out playback are impossible. If playback fails every time the limit is exceeded, the sequences will not play as expected.

A priority parameter is used to resolve this problem. Sounds set with a higher priority parameter value will have higher priority for playback. For example, if all channels are in use when the higher priority sound tries to play, other sounds that are playing will stop and the higher priority sound will play. By assigning certain sounds with a higher priority, these sounds will be played at a higher volume than sounds assigned a lower priority.

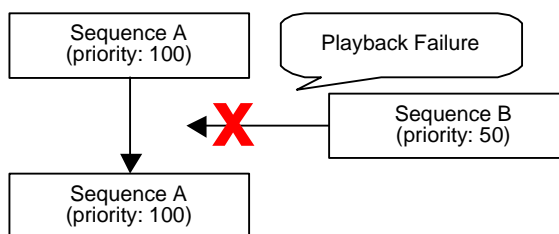
4.2 Player Priority

Player priority determines the priority of which sequences on a certain player should be played after the maximum number of sequences playing concurrently and the maximum number of channels have been reached.

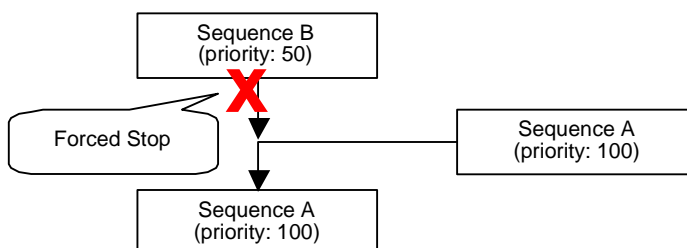
When a sequence is played, a check is first performed to determine the maximum number of sequences that can be played simultaneously on each player. If the maximum number of sequences is reached, the player priority value is compared to the lowest player priority value of the sequences playing on that player. As a result, the sequence with a greater player priority value will force the sequence with the lower player priority value to stop or end in a playback failure. If the player priority values of two sequences are the same, the sequence attempting to play is given priority over the sequence that is playing currently.

Figure 4-1 Example of Player Priority

Playing back sequence B when sequence A is being played back



Playing back sequence A when sequence B is being played back



Next, the system will check to see if the maximum number of sequences that can be played concurrently on the entire system is reached. If this limitation is reached, the priority is compared with the sequence that has the lowest player priority value of all the sequences that are being played concurrently. The sequence with the higher player priority value will force the sequence with the lower player priority value to stop or end in playback failure.

Sequences that are fading out will have the lowest player priority value of 0.

4.3 Voice Priority

4.3.1 What is Voice Priority?

Voice priority prioritizes the generation of a new sound when all 16 channels are in use.

The generation of the new sound and the currently playing sounds require seventeen voice priority values. These voice priority values are compared to each other and the sound with the lowest voice priority value is stopped. If the newly generated sound has the lowest voice priority value, the generation of the new sound will fail. If more than one playing sound has the lowest voice priority value, which sound will stop is indeterminate.

4.3.2 Setting Up Voice Priority

There are two types of parameters that are used to set up the voice priority values. One parameter is the sequence priority value that determines the priority of the entire sequence. The other parameter is the track priority value that determines the priority of each track in a sequence. These two parameters are combined to make up the final voice priority value.

For example, if the background music sequence voice priority is lowered, the sound effect sequence takes priority. Furthermore, if the track priority value for the background music, and the total voice priority value is set to a value that is greater than the voice priority value for a sound effect, the track that has the background music will be played without getting lost in the sound effects.

4.3.3 Release

A sound that is being released will be given the lowest priority regardless of the set parameters or priority values. As a result, sounds that are being released will fade out.

Sounds are released in the order of their current volume level. The sound will fade out starting with the sounds with the lowest volume level or the sounds closest to the end of their play time.

5 Memory Management

5.1 The Need for Memory Management

The sound data is stored in the DS ROM. However, the sound data cannot be used until it is loaded from the ROM into the main memory.

Because the capacity of main memory is small, not all of the sound data stored in the ROM can be loaded into main memory. Therefore, memory management loads the necessary sound data and then deletes unnecessary sound data. The following paragraphs outline the memory management system in NITRO-Composer.

A method called stream playback that does not require memory management can be used for sound generation. For details on stream playback, see Chapter 6 Stream Playback.

5.2 Two Types of Heap

Two types of heap are available in NITRO-Composer. Use these heaps according to their purpose.

- The sound heap
- The player heap

The sound heap is the main heap, and the player heap is part of the sound heap. Create the sound heap first, and then create the player heap, when it becomes necessary.

Below is a description of each heap.

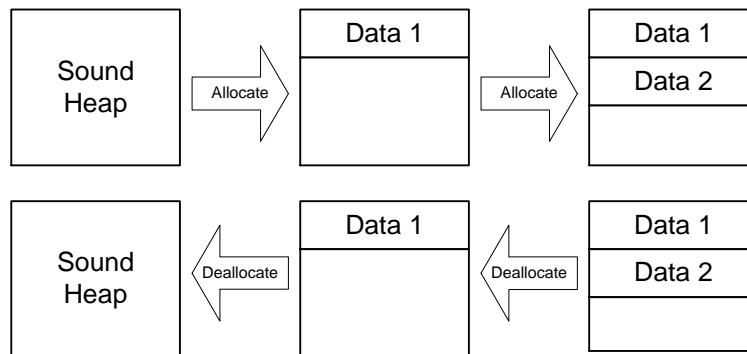
5.3 The Sound Heap

5.3.1 About the Sound Heap

The sound heap is a simple stack-based heap.

By loading the sound heap, data will be loaded from the top of the sound heap. Unnecessary data is deleted, in order, from the bottom of the heap.

Figure 5-1 Sound Heap Allocation and Deallocation



5.3.2 How to Use the Sound Heap

Data is loaded to and deleted from the sound heap during startup and between scenes.

Because loading large amounts of data is inefficient (for example, waveform data, frequently accessed data such as play time sound effect sequences), data can be loaded to the sound heap between scenes. Once the sound data has been loaded, the data can be used without reloading it until it is deleted.

5.3.3 Group Load

Sound data is loaded in sound data units or group units.

By loading group units, multiple sound data units can be loaded at once. For example, a group of resident sound data units can be created and loaded at startup.

The sound designer defines which sound data units each group will contain in the definition file of the sound archive.

5.4 The Player Heap

5.4.1 About the Player Heap

The player heap is where the data used during a sequence playback is stored.

If the needed data is not in the sound heap, the data is automatically loaded into the player heap at the time the sequence is played. The sequence is played using the data in the player heap. When sequence playback is complete, the player heap is deallocated.

For example, if there is bank and waveform data but no sequence data in the sound heap, the sequence is played after the sequence data is loaded into the player heap. After playback completes, the sequence data is deallocated from the player heap and the bank and waveform data remain in the sound heap. In this case, sequence data must be reloaded to play the same sequence again.

Figure 5-2 The Player Heap Instructions

Error! Not a valid link.

5.4.1.1 Cautions for Sequence Archive Playback

A sequence archive cannot be loaded into a player heap. The sequence archive must be loaded into the sound heap to play the sequence archive. The bank and waveform data are needed for sequence archive playback, but can be loaded into the player heap.

5.4.2 How to Use the Player Heap

Data loads automatically into the player heap when the sequence starts to play. If the loaded data set is too large or accessed by the player heap frequently, the loaded data will be a heavy processing load. Use the player heap primarily to load data such as background music sequences and bank data.

To avoid loading data onto the player heap, explicitly load data onto the sound heap in advance. If all the sequence playback data is in the sound heap, there is no need to create a player heap.

5.4.3 Creating the Player Heap

A player heap must be created for each player. To make multiple sequences play concurrently, the same number of player heaps must be created for the number of sequences played.

Generally, the sound designer configures the size of the player heap with the sound archive definition file. However, the programmer can also configure the size of the player heap.

The player heap is created by allocating memory from the sound heap. If a particular region of the player heap is deallocated, that player heap will no longer exist. The player heap can be recreated by reallocating the memory.

5.5 A Comparison of the Sound Heap and the Player Heap

Table 5-1 compares the characteristics and primary uses of the sound heap and player heap.

Table 5-1 A Comparison of the Characteristics and Primary Uses of the Sound and Player Heap

	The Sound Heap	The Player Heap
Load Timing	Primarily at startup and between scenes	When playing the sequence
Load Process	Call a load function	Automatically processed within the sequence play function
Deleting Data	Call a memory deallocation function	Automatically deleted when the sequence finishes playing
Primary Load Target	The waveform archive The sequence archive The bank data for sound effects	The sequence data The bank data for the background music
Advantages	Data loading and deletion can be controlled freely and processed efficiently. Use of Memory is also efficient because data is shared for all sequence playbacks.	Data loading and deletion is automatic so memory operations are unnecessary.
Disadvantages	Problems occur when the necessary data is unknown beforehand and when dealing with data that is only temporarily used.	Memory efficiency is poor because loading occurs during the game, so a player heap is needed for each playback sequence.

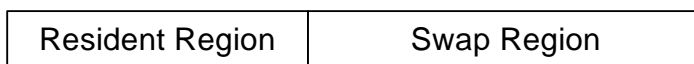
5.6 An Example of Memory Management

The following introduces a method to implement memory management.

5.6.1 Specifications

Divide the sound heap into two blocks as shown in Figure 5-3, the resident region and the swap region.

Figure 5-3 An Example of Memory Management



Data loads into the resident region at game startup, where the data is stored until game shutdown. The data used throughout the game, such as sound effects data, are stored here.

Data such as waveform data needed by the background music are stored in the swap region. The data stored in this region is deleted as scenes change.

The sequence data needed by the background music will also be loaded into the player heap.

5.6.2 Preparation

5.6.2.1 Creating the Group

The sound designer creates the groups that are stored in the resident and swap regions.

The resident region group is defined as group number 0. In this group, the sequence, bank, and waveform data for sound effects are registered.

Define three groups, Group 1, 2, and 3, for the groups in the swap regions. Load only one of these three groups based on the scene. Register the correct bank and waveform data to play the background music in each scene.

5.6.2.2 Configuring the Player Heap

Because the player heap stores the background music sequence data, create the player heap in the player designated for background music. The sound designer configures the player heap in the sound archive definition file.

Make the player heap large enough to store the background music sequence data files (*.sseq) that are played back.

5.6.2.3 Confirm the Configuration

The sound designer simulates any configuration problems in the NITRO-Player heap simulation mode or the SoundPlayer manual load mode. For further details, refer to the *NITRO-Player User Manual* or the *NITRO-Composer Sound Designer Guide*.

5.6.3 Startup Processing

5.6.3.1 Creating the Sound Heap

Create the sound heap first. Decide how much memory to use and allocate that memory for the sound heap.

5.6.3.2 Initializing the Sound Archive

To use the sound archive, first initialize the sound archive. Initializing the sound archive will fill the sound heap.

5.6.3.3 Creating the Player Heap

Create the player heap based on the sound archive setting. The memory for the player heap is allocated in the sound heap.

5.6.3.4 Loading Resident Groups

Load the resident groups. After loading the resident groups, sound effects can be played at any time.

5.6.4 Gameplay Processing

5.6.4.1 Loading to the Swap Region

Load the group for the swap region based on the scene. Bank and waveform data for the background music data are loaded.

5.6.4.2 Background Music Playback

When background music plays, the background music sequence data is loaded to the player heap. The bank and waveform data use the data in the sound heap to play the sequence.

5.6.4.3 Switching Scenes

Clear the swap region when switching scenes. Afterwards, reload the data to the swap region.

5.6.5 Memory Usage Conditions

After the processes above are carried out, the memory usage conditions are shown in Figure 5-4.

Figure 5-4 Memory Usage Conditions

Error! Not a valid link.

6 Stream Playback

6.1 What is Stream Playback?

Stream playback plays sound without loading all the sound data in advance. Instead, small chunks of data are stored in a buffer during playback time. Sequence playback using sequence data is different than stream playback.

With stream playback, a long sound from waveform data can be played using a minimum amount of RAM. As explained in Chapter 5, no memory management is required. However, stream playback has several restrictions that sequence playback does not.

The following sections explain the mechanism of stream playback and the differences between sequence and stream playback.

6.2 The Mechanism of Stream Playback

Stream playback is processed with the following steps.

1. Allocate channels
2. Prepare the data
3. Start playback
4. Load data as needed

6.2.1 Allocate Channels

Allocate the channels for stream playback. Two channels must be allocated for stereo output and one channel must be allocated for mono output.

Select channels that do not conflict with other channel functionalities that may be used at the same time. For example, if sound capture is used at the same time, channels 1 and 3 should not be used. For more information, see paragraph 2.3 Channel Numbers.

6.2.2 Prepare the Data

Before playback, the minimum chunk of data needed for playback must be loaded. Note that playback does not start until the chunk is completely loaded.

6.2.3 Start Playback

Start the actual playback.

6.2.4 Load Data as Needed

The chunks of data played back are deleted to make way for new data that is loaded for the next playback chunk. By repeating this process, playback can continue indefinitely.

6.3 Differences Between Stream and Sequence Playback

Stream and sequence playback each have advantages and disadvantages. Table 6-1 summarizes and compares the characteristics of sequence and stream playback.

Table 6-1 Characteristics of Sequence and Stream Playback

	Sequence Playback	Stream Playback
Data composition	Sequence data Bank data Waveform data	Waveform data only
Data loading	All data must be loaded before playback	Only a chunk of data is loaded automatically during playback
Playback processing load	Extremely light	Fairly heavy
Maximum simultaneous playbacks	Up to 16	Up to 4
Programmatic control	Can change many parameters, such as tempo and pitch	Cannot change any parameters, except volume and pan

Stream playback requires more processing resources and should not be used without evaluating circumstances. Up to four streams can play concurrently, but processing constraints make this difficult in practice.

Stream playback should be used when there is not enough memory for sequence playback or when the playback processing load is not a concern.

7 Output Effects

7.1 What are Output Effects?

This feature applies output effects, such as `surround`, to the sound output using sound capture. The following effect types are available.

Table 7-1 Output Effect Types

Type	Description
Normal	No effects applied to stereo output of the captured sound.
Surround	Distributes the sound across the DS system speakers.
Headphones	Reduces the stress on the ears when headphones are used.
Mono	Converts stereo sound to mono output.

7.2 Precautions About Using Output Effects

Pay particular attention to the following issues when using output effects.

7.2.1 Reduced Number of Simultaneous Sounds Generated

Normally, the DS can play 16 sounds concurrently. However, output effects use channel 1 and 3. If output effects are used, the DS can play a maximum of 14 sounds concurrently.

7.2.2 Processing Load

The processing load will be increased when surround and headphone effects are applied. The increase is approximately 5% of one frame process.

7.2.3 Sound Distortion

The sound is captured as a 16 bit signal, but the mixer processes it as a 24 bit signal. Therefore, layered sounds may be distorted.

The mixer processes the sound in 24 bits, but the sound is obtained with the sound capture in 16 bits.

7.3 Precautions when Using Surround

Pay particular attention to the following issues when using surround.

7.3.1 Increase in Volume

Because of the algorithms used for surround, the volume of each sound split into a left and right component is louder than normal stereo sound. Therefore, the final sound output is easily distorted. Furthermore, the overall volume balance is different than normal stereo sound. This difference is more noticeable when there are more sounds on one side.

7.3.2 Difference in Balance Between the Left and Right Components

With the DS stereo speaker, the left and right sounds may be slightly different. When using the surround effect, the difference between the left and right component may be emphasized. For example, a sound from far left may spread out differently from a sound from far right.

7.4 About Stereo and Mono Output

Normal and mono output can be implemented without the use of output effects. Normal output is when output effects are not used. Mono can be implemented using `NNS_SndSetMonoFlag`.

If only normal and mono output is used, not using output effects is preferable. However, if they are being used with sound, there are two choices. Each has its advantages and disadvantages.

7.4.1 When Using All Output Effects

Using output effects with normal and mono output will result in the types of limitations that are described in the paragraph 7.3 Precautions About Using Output Effects, including those in normal or mono output mode.

In contrast, the advantage is that the output sound is not much different from the output sound using surround or headphones. Also, switching the output effect type will not cause a loud noise while a sound is playing, since there is no need to halt the output effect.

7.4.2 When Using Only Surround and Headphone Output Effects

When using output effects only for surround or headphones, the problems that are described in paragraph 7.2 Precautions About Using Output Effects will not occur in a normal or mono output.

However, there are times when there are great differences between the sound when comparing surround or headphone output. Also, the output effects must be started or stopped when switching the output effect type, so the sound may be interrupted during sound generation.

8 Sound Processing

8.1 ARM7

Sound is mainly processed by the ARM7. The ARM7 is one of the CPUs included in the Nintendo DS system. Normal game and graphics are processed on a separate CPU called ARM9. The ARM7 performs wireless and microphone processes as well as sound processes.

8.2 Main Memory Access

Sound is processed on a different CPU than games and graphics. Therefore, the processing of these game components does not significantly affect their processing time. However, main memory access for sound requires some caution.

The main memory stores program, sound, and graphics data. Both ARM7 and ARM9 can access this data. If memory access is attempted simultaneously by both CPUs, ARM7 is given memory access priority over ARM9. When ARM7 accesses the main memory to process sound, ARM9 processes may stop.

There are two major processes that access the main memory when ARM7 processes sound.

- Waveform Data Transfer
- Sequence Data Processing

8.2.1 Waveform Data Transfer

The waveform data in the main memory transfers to the DS sound circuit and plays. The main memory is accessed when the transfer occurs.

The transfer rate increases when waveform data with a high sampling rate plays or the waveform data plays in a higher pitch than the original waveform data pitch. If these types of waveforms are played, watch the processing time.

8.2.2 Sequence Data Processing

When sequence data in the main memory is played, the main memory is accessed.

With sequence playback, a time delay between each sound generation generally occurs. During this time delay, the sequence processing stops so that the main memory is not accessed. For example, if the pitch continues to change between each sound generation, the main memory will be accessed each time.

8.3 Sound Processing Units

Sound processing is carried out in intervals of approximately 5.2 msec. A single interval is known as a sound frame. Sounds cannot be changed in intervals that are shorter than 5.2 msec.

8.3.1 Quarter Note Resolution

The quarter note resolution for the sequence data is 48. When the tempo is 240, exactly one tick (1/48 of a quarter note) will be equivalent to one sound frame.

© 2004-2007 Nintendo

The contents of this document cannot be duplicated, copied, reprinted, transferred, distributed or loaned in whole or in part without the prior approval of Nintendo.