TWL-System

NITRO-Composer Sound Tools Manual

2009/03/04

> **The content of this document is highly confidential and should be handled accordingly.**

Confidential

These coded instructions, statements, and computer programs contain proprietary information of Nintendo and/or its licensed developers and are protected by national and international copyright laws. 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

    3.3.5   DMA Transfer of Sound Data 20

4   SMF Converter: smfconv...21
  4.1   Overview..21
  4.2   Location of Executable File 21
  4.3   How to Use smfconv..21
    4.3.1   Syntax 21
    4.3.2   Execution Results 21
    4.3.3   Options...21
  4.4   Details...22
    4.4.1   Tracks 22
    4.4.2   Specifying Loops..22
    4.4.3   Blank Space at Start of SMF..22
    4.4.4   MIDI events..22

5   Sequence Converter: seqconv 23
  5.1   Overview..23
  5.2   Location of Executable File 23
  5.3   How to Use 23
    5.3.1   Syntax 23
    5.3.2   Execution Results 23
    5.3.3   Options...23

6   Bank Converter: bankconv 24
  6.1   Overview..24
  6.2   Location of Executable File 24
  6.3   How to Use 24
    6.3.1   Syntax 24
    6.3.2   Two Output Modes...24
    6.3.3   Waveform List File Output Mode 24
    6.3.4   Options...25
    6.3.5   Bank Conversion Mode 25

7   Waveform File Converter: waveconv 27
  7.1   Overview..27
  7.2   Location of Executable File 27
  7.3   How to Use 27
    7.3.1   Syntax 27
    7.3.2   Execution Results 27
    7.3.3   Options...27
  7.4   The Waveform Format...28

Tables

Figures

Revision History

Revision Date	Description
2009/03/04	Corrected error in the Japanese source document.
2008/05/30	Made revisions in line with the NITRO-System name change (updating to TWL-System)
2008/04/08	• Changed the format of the Revision History. • Changed page headers.
2007/03/14	• Added a description related to DMA transfer of sound data. • Added an option for specifying the alignment of sndarc. • Added an option for specifying the conversion filter type of sndarc. • Added an option for specifying the argument list file of bankconv.
2005/09/01	• Added an option for specifying an output filename to sndarc. • Added an option for specifying a preprocess file to sndarc.
2005/01/31	• Added a description related to the `#include<file>` format. • Added the `-I` option to `seqconv` and `bankconv`. • Changed "NITRO" to "Nintendo DS."
2004/10/12	• Added description about the limitation on the loop starting location of `waveconv`.
2004/09/16	• Revised the description of `*.sadl` and `*.sbdl` files that are output by `sndarc`. • Unified `*.sadl` file names to "sound label files." • Name of the Bank list file (`*.sbdl`) was changed to "sound archive label files."
2004/09/02	• Added description regarding the bank list file output by sndarc. • Changed the example of `#include`.
2004/08/10	• Added `strmconv`.
2004/07/20	• Added `the -b` option `sndarc`. • Changed the specifications of `bankconv`. • Added a description of loop correction in `waveconv`. • Style adjustments.
2004/06/01	• Changed the `#include` example to a `*.spdl` file. • Added "Execution Results" heading to the description of each converter. • Added and revised description of list files created by `sndarc`, `seqconv` and `bankconv`.
2004/04/12	Initial version.

1 Introduction

This document provides a detailed explanation of the sound tools for sound designers. The intended audience is sound designers who have experience generating sound data using NITRO-Composer. If you are new to using NITRO-Composer, read the Sound Designer Guide.

1.1 The Structure of This Document

This document is composed of independent chapters, which can be read in any order. The following sections summarize each chapter.

1.1.1   Chapter 2 Preprocessor Directives

This chapter explains the preprocessor directives, which can be used to support descriptions of text in all text-format sound data files.

1.1.2   Chapter 3 Sound Archiver: sndarc

This chapter explains the command-line syntax for sndarc, the sound archiver.

1.1.3   Chapter 4 SMF Converter: smfconv

This chapter explains the command-line syntax for smfconv, the SMF converter.

1.1.4   Chapter 5 Sequence Converter: seqconv

This chapter explains the command-line syntax for seqconv, the sequence converter.

1.1.5   Chapter 6 Bank Converter: bankconv

This chapter explains the command-line syntax for bankconv, the bank converter.

1.1.6   Chapter 7 Waveform File Converter: waveconv

This chapter explains the command-line syntax for waveconv, the waveform converter.

1.1.7   Chapter 8 Waveform File Archiver: wavearc

This chapter explains the command-line syntax for wavearc, the waveform archiver.

1.1.8   Chapter 9 Stream Converter: strmconv

This chapter explains the command-line syntax for strmconv, the stream converter.

2 Preprocessor Directives

2.1 Overview

The preprocessor directives are commands that define and interpret text descriptions in text-format sound data files.

An example of a preprocessor directive is `#define`, as shown below.

```
#define LENGTH 24
```

Using this directive, `LENGTH` can be substituted for value assignments (such as 24) in the following lines. For example, if there are lines throughout the code that use `LENGTH`, all instances of `LENGTH` can be changed by changing the assigned value of `LENGTH` in the `#define` line. This assignment is useful for determining correct values by trial and error without changing every variable assigned the same value.

2.2 The Preprocessor Directives

A list of the preprocessor directives is shown in Table 2-1.

Table 2-1 List of Preprocessor Directives

Preprocessor Directives	Explanation	Ref.
`#define`	Defines the replacement macros.	2.3
`#undef`	Disables a replacement macro definition.	2.4
`#include`	Includes external files.	2.5
`#if`	Evaluates a TRUE-FALSE condition.	2.6
`#ifdef`	Evaluates a defined-macro condition.	2.7
`#ifndef`	Evaluates an undefined-macro condition.	2.7
`#else`	Indicates the beginning of statements to execute if condition is FALSE for `#if, #ifdef, #ifndef, or #elif`.	2.8
`#endif`	Indicates the end of a segment that starts with `#if, #ifdef, #ifndef,` or `#elif`.	2.9
`#elif`	Evaluates condition on a FALSE returned from `#if, #ifdef,` or `#ifndef`.	2.10

These brief descriptions may be difficult to understand, but the referenced sections thoroughly explain each topic.

2.3 #define

`#define` defines a replacement macro and is written in the following format.

```
#define macro_name replacement_value
```

If subsequent character strings are the same as `macro_name`, those character strings are replaced with `replacement_value`. The character string must exactly match `macro_name` because `macro_name` is case-sensitive.

Similar to label names, macro names are specified by strings that begin with a letter, followed by letters, numerals, and underscore (`_`) characters. Normally, macro names do not use lowercase letters.

Any data type can be used as replacement values, but each replaced value must be of the same type as the defined macro name. For example, if a macro is defined as an `int` type, the replaced value must also be of `int` type. With some directives, the `replacement_value` can be omitted, such as when you use `#ifdef`.

An error will occur if the same macro name is defined twice. The same macro name can be redefined by first using `#undef` to disable the first definition.

2.3.1 Examples

The following code uses `#define` to define a value of 24 for `LENGTH`.

```
#define LENGTH 24

  cn4 127, LENGTH
  dn4 127, LENGTH
  en4 127, LENGTH
  fin
```

The following code provides the same result.

```
  cn4 127, 24
  dn4 127, 24
  en4 127, 24
  fin
```

Replacement values can be described in various ways. The following code will have the same result as the two previous code examples.

```
#define VEL_LEN 127, 24

  cn4 VEL_LEN
  dn4 VEL_LEN
  en4 VEL_LEN
  fin
```

The following is an example combining #define and #indef.

```
#define ENABLE_FLAG

  cn4 127, 24
  dn4 127, 24
#ifdef ENABLE_FLAG
  en4 127, 24
#endif
  fin
```

In this example, the line of code below #ifdef executes only when ENABLE_FLAG is defined. If [#define ENABLE_FLAG] is deleted, [en4 127, 24] becomes disabled.

2.4 #undef

#undef disables a replacement value previously defined by #define. It is written in the following format.

```
#undef macro_name
```

When #undef is used, the previously defined macro no longer exists and can be redefined using #define. When an unassigned macro name is assigned to a variable in a program, nothing happens.

Use the #undef command to redefine a macro or to use the macro name with #ifdef or #ifndef.

2.4.1 Examples

```
#define LENGTH 24

  cn4 127, LENGTH
  dn4 127, LENGTH
  en4 127, LENGTH

#undef LENGTH
#define LENGTH 12

  cn4 127, LENGTH
  dn4 127, LENGTH
  en4 127, LENGTH
  fin
```

In the previous code example, #undef is used to make a new definition for a macro of the same name.

```
#define ENABLE_FLAG
#undef ENABLE_FLAG

  cn4 127, 24
  dn4 127, 24
#ifdef ENABLE_FLAG
  en4 127, 24
#endif
  fin
```

In the previous example, the line after #ifdef is valid only when ENABLE_FLAG is defined. Since the replacement macro is disabled by #undef, [en4 127, 24] is ignored.

2.5 #include

`#include` includes external files and is written as follows.

```
#include "filename"
#include<filename>
```

The file that is specified by `filename` is read into the current file.

Using `#include`, code that is shared by different files can be written in a single file, and a single file can be divided into more files so that sections of code can be edited by different people.

Either an absolute path or a relative path can be used in the filename. If a relative path is designated, the source directory will depend on the two formats. When using the `#include "filename"` format, it defines a relative path from the current file. If `#include<filename>` is used, it defines a relative path from the include path that has been registered.

The include path can be designated by using the converter command line option `-I`. If multiple include paths are designated, they are searched for in the order in which they were registered. When using sndarc, the directory that contains the sound archive definition file (`*.sarc`) is registered as the include path. Therefore, the `#include<filename>` format can be thought of as the relative path from the directory that contains the sound archive definition file (`*.sarc`).

2.5.1 Examples

If macro names are assigned to the instruments in the bank definition file, the program number list file (that has the SPDL extension) looks similar to the following code.

```
#define PRG_PIANO 0
#define PRG_ORGAN 1
#define PRG_GUITAR 2
```

If this file is placed in an `#include` statement at the top of a text sequence, macro names from the program number list file can be used in the following lines of code.

```
#include "../bnk/se.spdl"

  prg PRG_ORGAN
  cn4 127, 24
  fin
```

To combine code into one file, see Figure 2-1. To see how to divide a file so that different people can edit one file, see Figure 2-2.
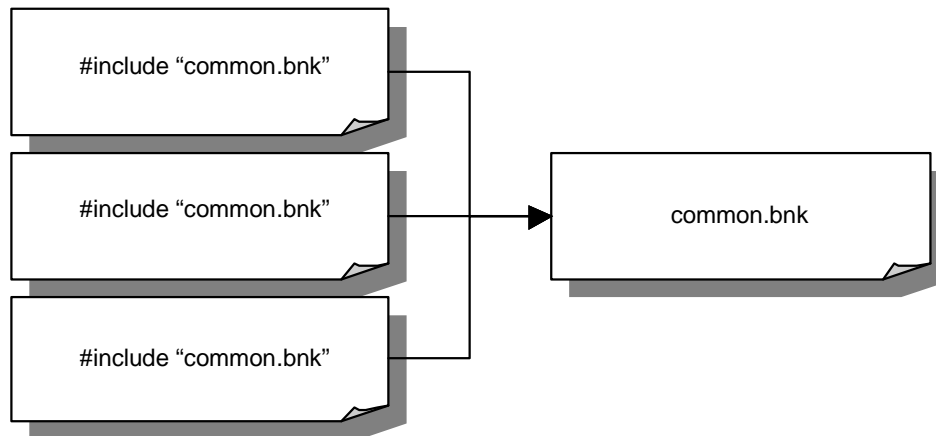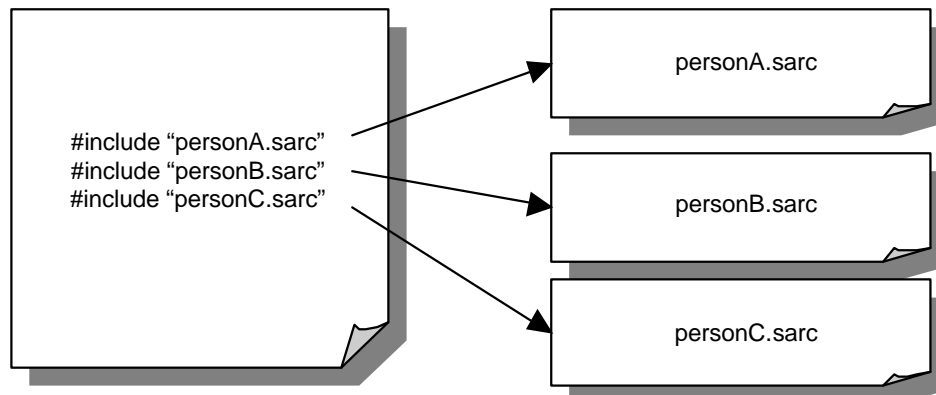
Figure 2-1 Extracting a Shared Part

Figure 2-2 Dividing a File for Editing by Multiple People

2.6 #if

The #if directive evaluates a TRUE-FALSE condition. An example of #if is written as follows.

```
#if evaluation expression
   (Valid if TRUE)
#endif
```

If the result of the evaluation expression is TRUE, all the lines up to #endif become valid. Otherwise, those lines are disabled. The #endif marks the end of the code that is executed when the condition is TRUE.

Numeric values can be specified for this evaluation expression. When the numeric values are specified, zero signifies *disabled*, and non-zero signifies *enabled*.

2.6.1 Examples

A section of code can be commented out by disabling it, as shown below.

```
#if 0
cn4 127, 24
  dn4 127, 24
  en4 127, 24
  fin
#endif
```

Using this approach, it is easy to enable the section at a later time.

```
#if 1
cn4 127, 24
  dn4 127, 24
  en4 127, 24
  fin
#endif
```

You can use evaluation expressions to switch lines on and off.

```
#define VERSION 2

cn4 127, 24
  dn4 127, 24
#if VERSION >= 2
  en4 127, 24
#endif
  fin
```

By evaluating the expression as shown in the code examples, the section can be enabled or disabled. In the example above, the result of the evaluation expression is true, so the line [en4 127, 24] is enabled.

2.7 #ifdef / #ifndef

#ifdef and #ifndef evaluate whether a macro has been defined. These directives are written in the following format.

```
#ifdef macro_name
#ifndef macro_name
```

Specify the macro name instead of an evaluation expression to check whether the macro name is defined. #ifdef becomes TRUE if a macro is defined, and #ifndef becomes true if the macro is undefined. Besides checking whether a macro is defined, they behave the same as #if.

2.7.1 Examples

```
#define ENABLE_FLAG

  cn4 127, 24
  dn4 127, 24
#ifdef ENABLE_FLAG
  en4 127, 24
#endif
#ifndef ENABLE_FLAG
  fn4 127, 24
#endif
  fin
```

In the previous example, the `ENABLE_FLAG` macro is defined, so the line `[en4 127, 24]` is enabled and the line `[fn4 127, 24]` is disabled.

```
#define ENABLE_FLAG
#undef ENABLE_FLAG

  cn4 127, 24
  dn4 127, 24
#ifdef ENABLE_FLAG
  en4 127, 24
#endif
#ifndef ENABLE_FLAG
  fn4 127, 24
#endif
  fin
```

On the other hand, if `#undef` is added as shown above, `ENABLE_FLAG` becomes undefined and the line `[en4 127, 24]` is enabled.

2.8 #else

If `#if, #ifdef, #ifndef,` or `#elif` evaluates to FALSE, `#else` marks the beginning of the code that executes when the FALSE condition is met. The general expression is similar to the following code.

```
#if ...
   (Valid if TRUE)
#else
   (Valid if FALSE)
#endif
```

When the evaluation result of `#if` or `#ifdef` is FALSE, the code after `#else` and before `#endif` is enabled. Conversely, if the result is TRUE, this section is disabled.

2.8.1 Examples

By writing the code as shown below, the code can be switched easily from TRUE to FALSE.

```
#if 1
cn4 127, 24
  dn4 127, 24
  en4 127, 24
  fin
#else
cn4 127, 24
  en4 127, 24
  gn4 127, 24
  fin
#endif
```

In this example, the section below `#if` and above `#else` is valid. If you change the 1 to 0, the other section is enabled.

2.9 #endif

The `#endif` directive indicates the end of the evaluation segment that started with `#if`, `#ifdef`, `#ifndef`, or `#elif`. The general expression is similar to the following code.

```
#if evaluation expression
  (Valid if TRUE)
#endif
```

For details, read the explanations of `#if`, `#ifdef`, `#ifndef`, and `#elif`.

2.10 #elif

The `#elif` directive defines a second expression to evaluate if the result of the initial `#if`, `#ifdef`, or `#ifndef`. directive is FALSE. The general expression is similar to the following code.

```
#if evaluation expression 1
#elif evaluation expression 2
#endif
```

The best way to explain this directive is to look at the following examples.

2.10.1 Examples

```
#define VERSION 2

#if VERSION == 1
cn4 127, 24
#else
#if VERSION == 2
dn4 127, 24
#else
en4 127, 24
#endif
#endif
  fin
```

The code below is more condensed than the code above when `#elif` is used.

```
#define VERSION 2

#if VERSION == 1
cn4 127, 24
#elif VERSION == 2
dn4 127, 24
#else
en4 127, 24
#endif
  fin
```

3 Sound Archiver: sndarc

3.1 Overview

The sound archiver, sndarc, is a command line tool that combines a number of sound data files into a single file. It can also convert various types of sound data simultaneously. When converting the files, the file timestamps are compared so that only the necessary files are converted.

Normally, when `MakeSound.bat` is executed in the SoundPlayer development environment, the sound archiver runs automatically. Therefore, there is no need to start the tool manually. However, if you want to configure the option settings explained in the following sections, you need to sndarc from the command line.

3.2 Location of Executable File

The sndarc executable file `sndarc.exe` is located in `$TwlSystem/tools/win/bin`. To convert the various types of sound data, use `seqconv.exe`, located in the same directory.

3.3 How to Use sndarc

3.3.1 Syntax

Command line statements and arguments use the following syntax.

```
sndarc [options] <inputfile>
```

For `<inputfile>`, specify a sound archive definition file.

3.3.2 Execution Results

When the conversion process is executed, a sound archive file changes the file extension from the `<inputfile>` file extension to SDAT and outputs the file.

The following output files are produced.

- Sound label file (`*.sadl`)
- Sound archive label file (`*.sbdl`)
- Sound map file (`*.smap`)

It is possible to change these filenames through the use of options.

3.3.2.1 Sound Label List File

A sound label file (with the extension SADL) contains `#define` index numbers for labels defined in the sound archive. This file also includes the contents of the sequence archive label file (with the extension SSDL) that is output during the sequence archive conversion process.

If this file is included with `#include` in a program, the labels that are defined in the sound data can be used in that program.

Since this file is generated after various types of data are converted, it is not possible to use `#include` in the sound data text to load this file.

3.3.2.2   Sound Archive Label File

The sound archive label file (with the extension SBDL) uses `#define` to define index numbers for labels in the sound archive. This file is generated before other types of data are converted so that it can be loaded from the sequence archive text file by using `#include`.

3.3.2.3   Sound Map File

A sound map file (with the extension SMAP) is a text file containing information about the type of data that is stored in the sound archive.

3.3.3 Options

The following options are available.

Table 3-1 List of sndarc Options

Options	Explanation
`-c`	Ignores the timestamp of the file and converts all files.
`-b`	Does not output symbol data to the sound archive.
`-s, --silent`	Does not show internal executing commands.
`-v, --verbose`	Shows internal activity in detail.
`-h, --help`	Displays help.
`-o, --sdat filename`	Specifies the sound archive (`*.sdat`) and sound map (`*.smap`) filename.
`--sadl filename`	Specifies the filename of a sound label file (`*.sadl`).
`--sbdl filename`	Specifies the filename of a sound archive label file (`*.sbdl`).
`-f filename`	Specifies a preprocessed file. Preprocessed files are processed before the sound definition file that has been specified as an argument. If more than one file was specified, they are processed in the order of specification.
`--align byte`	The sound data within the sound archive is aligned at the specified number of bytes.
`--convert flags`	Specifies the file type on which to perform conversion (details follow).

3.3.4 Conversion File Types

Specifying the `--convert` option makes it possible to specify the file type on which to perform conversion. For example, with `--convert s`, sequence archive files are converted, but other bank files, stream files, and so on are not converted. If you only want to convert sequence archive files, for

example, specifying an option such as this can decrease the conversion time.

The file types that can be specified are shown in Table 3-2.

Table 3-2 Conversion File Types

File Type	Description
a	All file types (default).
q	Sequence files.
s	Sequence archive files.
b	Bank files.
w	Waveform archive files.
m	Stream files.
n	Do not convert any file types.

It is also possible to specify multiple file types, such as `--convert sqm`.

Note that specifying not to convert files that have changed may cause illegal data to be output.

3.3.5 DMA Transfer of Sound Data

When loading sound data within a sound archive, the offset and size of the data to be loaded must be 512-byte aligned if you want to use DMA transfer. Normally, the offset is 32-byte aligned and the size is the file size of the actual data. The offset and size can be aligned to 512-bytes by specifying the `--align` option.

However, please note the following issues.

3.3.5.1   Offsets of Sound Archives

The offset of the sound archive itself must be aligned to 512 bytes separately from the `--align` option.

3.3.5.2   Data to Be Aligned

The `--align` option is valid for the following types of sound data.

- Stream data (`*.strm`)
- Sequence data (`*.sseq`)
- Sequence archive data (`*.ssar`)
- Bank data (`*.sbnk`)
- Waveform archive data (`*.swar`)

There is no effect on waveform data (`*.swav`), so when using the feature to load individual waveform data files, note that DMA transfer cannot be performed.

4 SMF Converter: smfconv

4.1 Overview

The SMF converter converts format 0 or format 1 Standard MIDI File (SMF) into a text format sequence file. The sequence file that is output in text format can be converted to a binary file with seqconv.

Under normal circumstances, the SMF converter is called automatically from the sound archiver, so there is no need to manually call smfconv.

4.2 Location of Executable File

The executable file `smfconv.exe` is located in `$TwlSystem/tools/win/bin`.

4.3 How to Use smfconv

4.3.1 Syntax

The command-line statement and arguments use the following syntax.

```
smfconv [options] <inputfile>
```

Specify an SMF for `<inputfile>`.

4.3.2 Execution Results

Executing the conversion process outputs a file with the extension SMFT changed from file extension `<inputfile>`.

4.3.3 Options

The options are listed in Table 4-1.

Table 4-1 List of smfconv Options

Options	Explanation
-o <filename>	Specifies output file name.
-v, --verbose	Shows internal activities in detail.
-u, --update	Converts only when input file has been updated.
-h, --help	Displays Help.

4.4 Details

4.4.1 Tracks

A maximum of 16 tracks can be used. Channel 1 to 16 corresponds to tracks 0 to 15. MIDI events like tempo changes that affect the overall sequence get mixed into track 0 for output.

4.4.2 Specifying Loops

To loop all tracks with the same timing, use the MIDI sequencer to insert square brackets as markers in the SMF. The opening bracket ([ ) defines the start of the loop and the closing bracket ( ]) defines the end of the loop. The section between the brackets is converted as a loop.

4.4.3 Blank Space at Start of SMF

When the SMF is converted, all blank spaces up to the first Note On are automatically cut. To prevent this from occurring, add a low-volume dummy note to the start of the sequence.

4.4.4 MIDI events

To learn about the MIDI events that are converted, see the list of sequence commands in the *Sequence Data Manual*.

5 Sequence Converter: seqconv

5.1 Overview

The sequence converter is the command-line tool that converts text-format sequence files into binary files. It can convert SMFT-format text files output by the SMF converter and the sequence files in the sequence archives format.

Normally, this sequence converter is called automatically from the sound archiver, so there is no need to manually call `seqconv`.

5.2 Location of Executable File

The executable file `seqconv.exe` is located in `$TwlSystem/tools/win/bin`.

5.3 How to Use

5.3.1 Syntax

The command-line statement and arguments use the following format.

```
seqconv [options] <inputfile>
```

For `<inputfile>`, specify a file that is in text format. To convert files in sequence archive format, the `--archive` option must be specified.

5.3.2 Execution Results

Executing the conversion process outputs a file with the extension `<inputfile>` changed to SSEQ. Converting a sequence archive outputs a file with the file extension changed to SSAR.

Converting a sequence archive also outputs a sequence archive label file with an SSDL extension. This file uses `#define` to define the sequence label set as an index number in the sequence table.

5.3.3 Options

Options are listed in Table 5-1.

Table 5-1 List of seqconv Options

Options	Explanation
-a, --archive	Converts sequence archive.
-o <filename>	Specifies output file name.
-I <dir>	Specifies the include path.
-v, --verbose	Shows internal actions in detail.
-u, --update	Converts only when input file has been updated.
-h, --help	Displays Help.

6 Bank Converter: bankconv

6.1 Overview

The bank converter is the command-line tool that converts the text-format bank definitions files into binary. Executing bank conversion also converts AIFF and WAV files that are registered in bank files. For the conversion process, the timestamp of files can be compared to only execute as needed.

Normally, the bank converter is called automatically from the sound archiver, so there is no need to call bankconv manually.

6.2 Location of Executable File

The executable file `bankconv.exe` is located in `$TwlSystem/tools/win/bin`. Waveform files are converted using `waveconv.exe` in the same directory.

6.3 How to Use

6.3.1 Syntax

The command-line statement and arguments use the following format.

```
bankconv [options] <inputfile>
```

Specify a bank definition file for `<inputfile>`.

6.3.2 Two Output Modes

The bank converter runs in two modes. The first mode is called *waveform list file output mode*, which extracts the waveform file information to output one waveform list file. The other mode is the *bank conversion mode*, which outputs the bank definition file as a bank binary file.

Specify option `-l` to use the Waveform List File Output Mode. The Bank Conversion Mode is the default mode.

6.3.3 Waveform List File Output Mode

6.3.3.1 Waveform Group Number

Specify the waveform group number following the input file name as shown below.

```
bankconv -l bank1.bnk:2 bank2.bnk:1
```

If only a filename is specified, the waveform group number is 0. The waveform group number is a value specified by `@WGROUP` in the bank definition file. It extracts only the waveform files that are registered in the specified waveform group.

Note: If `@WGROUP` is not specified, all waveform files are registered in waveform group 0.

6.3.3.2   Execution Results

The waveform file information is extracted from multiple bank definition files and output as a waveform list file. The waveform list file name must be specified with the `-o` option. The generated waveform file can be used in bank convert mode or with the waveform archiver.

Extracted waveform files can be converted with `waveconv`.

6.3.4   Options

Options are listed in Table 6-1.

Table 6-1 List of bankconv Waveform List File Output Mode

Options	Explanation
`-l`	Uses waveform list file output mode (required).
`-c`	Ignores the timestamp of the file and converts all waveform files.
`-o <filename>`	Specifies output filename (required).
`-I <dir>`	Specifies the include path.
`-s, --silent`	Does not show internally executing commands.
`-v, --verbose`	Shows internal activities in detail.
`-u, --update`	Converts only when input file has been updated.
`-h, --help`	Displays Help.

6.3.5   Bank Conversion Mode

6.3.5.1   Execution Results

Running a conversion outputs a file with the filename extension of the `<input file>` changed to SBNK. To use waveform files, you need to specify the waveform list file that supports each waveform set using the options `--wave0` to `--wave3`.

The bank conversion outputs a program number list file (with the file extension SPDL). This file uses `#define` to define a program number for the label set for each instrument.

6.3.5.2   Options

Options are explained in Table 6-2.

Table 6-2 List of bankconv Bank Conversion Mode Options

Option	Description
`--wave0 <filename>`	Specifies the waveform list file for waveform set 0.
`--wave1<filename>`	Specifies the waveform list file for waveform set 1.
`--wave2 <filename>`	Specifies the waveform list file for waveform set 2.
`--wave3 <filename>`	Specifies the waveform list file for waveform set 3.
`-o <filename>`	Specifies output file name.
`--arglist <filename>`	Writes the input file name in the specified file and passes it.
`-I <dir>`	Specifies the include path.
`-s, --silent`	Does not show internally executing commands.
`-v, --verbose`	Shows internal activities in detail.
`-u, --update`	Converts only when input file has been updated.
`-h, --help`	Displays Help.

7 Waveform File Converter: waveconv

7.1 Overview

The waveform file converter s the command line tool that converts AIFF and WAV files into SWAV files, the special NITRO-Composer waveform file format.

Normally, this waveform file converter is called automatically from sndarc, so there is no need to call `waveconv` manually.

7.2 Location of Executable File

The executable file `waveconv.exe` is located in `$TwlSystem/tools/win/bin`.

7.3 How to Use

7.3.1 Syntax

The command-line statements and arguments use the following format.

```
waveconv [options] <inputfile>
```

Specify a waveform file for `<inputfile>`.

7.3.2 Execution Results

Running the conversion process outputs a file with the `<inputfile>` extension changed to SWAV.

7.3.3 Options

Options are listed in Table 7-1.

Table 7-1 List of waveconv Options

Options	Explanation
`-s, --pcm16`	Converts into 16-bit PCM format.
`-c, --pcm8`	Converts into 8-bit PCM format.
`-a, --adpcm`	Converts into ADPCM format.
`-o <filename>`	Specifies output file name.
`-v, --verbose`	Shows internal activities in detail.
`-u, --update`	Converts only when input file has been updated.
`-h, --help`	Displays Help.

7.4 The Waveform Format

7.4.1 File Formats

Two waveform file formats are supported: Audio Interchange File Format (AIFF) and Wave (Microsoft).

7.4.2 Number of Channels

The number of channels used must be 1 or mono.

7.4.3 Quantifying Bit Number

The quantifying bit number is changed automatically to the output format by the converter so there are no real restrictions. However, it is advised that you set to the output format in advance. For example, set 8-bit sound to output as 8-bit PCM, and 16-bit sound to output in ADPCM.

7.4.4 Sampling Rate

There are almost no restrictions on the sampling rate, but the recommended range is 4 kHz to 44.1 kHz. The higher the sampling rate, the better the sound quality. However, higher sampling rates cause data transfer to occur more frequently and cause the processing load to increase.

7.4.5 Original Key

Describe the original key in the bank file rather than setting it in the waveform file. If the original key is set in the waveform file, the setting is ignored.

7.4.6 Loops

One type of loop is supported.

7.5 Details

7.5.1 Loop Correction

The waveform converter uses loop correction during conversion.

The hardware specifications of the TWL and Nintendo DS system require that loop lengths be multiples of a certain number (8 for ADPCM). Because it is inconvenient to set the loop length during the waveform data creation, the sampling rate is converted so that the loop length is set to multiples of that value during conversion.

For example, if the loop length is 500, the length will be changed to 504, a multiple of 8. Therefore, the sampling rate is multiplied by 1.008 (or 504/500). The size of the waveform data will increase accordingly.

7.5.2 Limitation of Loop Starting Point

Due to TWL and DS hardware specifications, the loop starting point cannot be moved back significantly.

The limitation of the loop starting point varies depending on the format as shown in Table 7-2. Also note that these limitation values indicate the value after loop normalization.

Table 7-2 Limitation of Loop Starting Point

Format	Maximum Value [sample]
ADPCM	524272
16-bit PCM	131070
8-bit PCM	262140

If the loop starting point is set beyond the limit, the conversion results in an error.

8 Waveform File Archiver: wavearc

8.1 Overview

The waveform file archiver `wavearc` is the command line tool that combines multiple waveform files into a single file, based on the waveform file list. The waveform converter `waveconv` outputs SWAV format files.

Normally, this waveform file archiver is called automatically from the sound archiver `sndarc`, so there is no need to call `wavearc` manually.

8.2 Location of Executable File

`wavearc.exe` is located in `$TwlSystem/tools/win/bin`.

8.3 How to Use

8.3.1 Syntax

The command line statement and arguments look like this.

```
wavearc [options] <inputfile>
```

Specify the waveform file list for `<inputfile>`, which is a list of waveform file names that has one file name per line written in text format.

Running the conversion process outputs a file with the `.swar` extension changed from the `<inputfile>` extension.

8.3.2 Options

Options are listed in Table 8-1.

Table 8-1 List of wavearc Options

Options	Explanation
-o <filename>	Specifies output filename.
-v, --verbose	Shows internal actions in detail.
-u, --update	Converts only when input file has been updated.
-h, --help	Displays Help.

9 Stream Converter: strmconv

9.1 Overview

Stream converter is the command-line tool that converts AIFF and WAV files to STRM files. The stream file format is used specifically by NITRO-Composer.

Normally, this tool is called automatically from sndarc so there is no need to call strmconv manually.

9.2 Location of Executable File

The executable file `strmconv.exe` is located in `$TwlSystem/tools/win/bin`.

9.3 Using the Tool

9.3.1 Syntax

The command-line statement and arguments use the following format.

```
strmconv [options] <inputfile>
```

Specify a waveform file for `<inputfile>`.

9.3.2 Execution Results

Running the conversion process outputs a file with the `<inputfile>` extension changed to STRM.

9.3.3 Options

Options are listed in Table 9-1.

Table 9-1 List of strmconv Options

Options	Explanation
-s, --pcm16	Converts into 16-bit the PCM format.
-c, --pcm8	Converts into 8-bit the PCM format.
-a, --adpcm	Converts into the ADPCM format.
-o <filename>	Specifies the output filename.
-v, --verbose	Shows the internal activities in detail.
-u, --update	Converts only when the input file has been updated.
-h, --help	Displays Help.

9.4 The Waveform Format

9.4.1 File Formats

Two waveform file formats are supported: Audio Interchange File Format (AIFF) and Wave (Microsoft).

9.4.2 Number of Channels

Any number of channels can be used.

9.4.3 Quantifying Bit Number

The converter automatically converts the quantifying bit number to the output format, so there are no restrictions. However, it is suggested that you set the output format in advance. For example, set 8-bit sound to output as 8-bit PCM, or 16-bit sound to output as ADPCM.

9.4.4 Sampling Rate

The converter puts no restrictions on the sampling rate. The higher the sampling rate, the higher the sound quality. However, data transfer occurs more frequently, increasing the processing load.

9.4.5 Loops

One type of loop is supported.

9.5 Resampling

The stream converter may resample the data during conversion.

NITRO-Composer cannot play streams at particular sampling rates. This unfixed sampling rate complicates the waveform data creation process, so the sampling rate is changed during conversion. Therefore, the size of the data after the conversion will increase slightly.

All company and product names in this document are the trademarks or registered trademarks of the respective companies.