NintendoWare for CTR

Sequence Data Manual

2010/11/04

Ver. 1.5.0

PROVISIONAL TRANSLATION

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

Table of Contents

Code

Tables

Figures

1 Introduction

This manual describes how to create sequence data used by SoundMaker.

1.1 What Is Sequence Data?

Sequence data corresponds to the music data represented in a standard MIDI file (referred to as “SMF” in this document). Commands that process changes in attributes such as sound generation and volume change over time are described. Since sequence data is simply a set of commands, sounds are actually produced using bank data that corresponds to the sound source data.

1.2 Text Sequences and Standard MIDI Files

The sequence data used by SoundMaker can be classified into two main types.

The first type is when a single sequence (track) is contained in a single sequence data file such as an SMF. The second type is when more than one set of sequence data has been gathered into a single file.

Both of these types of sequence data files are supported under SoundMaker using files called text sequences (extension `.cseq`).

SMF belongs to the first type of sequence data. At the time of conversion, it is converted into text sequences using sequence data belonging to the first type. It is then possible to create sequence data of the second type by directly editing the text sequences with a text editor.

This manual describes how to create these two types of sequence data.

2 Basic Common Features of Sequence Data

2.1 Time Base

The default value for time base (at quarter-note resolution) is 48. The time units for the time base scale are called ticks. In other words, the default length of a quarter note is 48 ticks.

The default tempo is 120.

2.2 Comments

You can include comments in text sequences. Under the conventions used for text sequences, when a semi-colon (;) is encountered in a line, everything from that point to the end of the line (line break) is treated as a comment.

Code 1 Examples of Comments

```
;;; comment
Track_0:  ; comment
```

2.3 Labels

A colon (:) placed after a character string in a text sequence file is treated as a label definition. It is possible to specify a jump from one location to another by defining such labels.

If a text sequence containing more than one set of sequence data is used under SoundMaker, labels are used to specify the playback starting position.

2.3.1 Characters That Can Be Used in Labels

Label names must begin with an alphabetic character. Each subsequent character must be a single-byte alphanumeric character or an underscore (_).

Code 2 Examples of Labels

```
LoopStart:
test_seq:
Track_01
```

2.3.2 Local Labels

If an underscore is used as the first character in a label name, it will be treated as a local label.

A local label is only valid between other labels that are not local labels. The same local label name can therefore be used in a different section.

Code 3 Examples of Labels and Local Labels

```
label_1:
  ;; local on the third line can be used here
_local:
  ;; local on the third line can be used here

label_2:
  ;; local on the seventh line can be used here
_local:
  ;; local on the seventh line can be used here

label3:
  ;; local cannot be used here
```

3 Standard MIDI Files

This section describes how to create sequence data from a standard MIDI file.

At the time of conversion, the standard MIDI file is converted into a text sequence file at time of conversion. The text sequence file is written using sequence commands described later in this document.

3.1 Format

Format 0 or Format 1 can be used as the SMF format. Format 2 is not supported.

3.2 Tracks

Up to 16 separate tracks can be used. However, there is a limit on the number of tracks that can be used by the overall nw::snd (NW4C sound runtime library) system.

Channel numbers 1 through 16 correspond to track numbers 0 through 15, respectively. In addition, MIDI events that affect the entire sequence, such as tempo change, can be mixed in and output on Track 0.

3.3 Looping an Entire Sequence

To loop all tracks in an SMF under the same timing, insert square brackets (`[ , ]`) as markers on the MIDI sequencer. The position of the opening bracket (`[`) is taken as the start point and the position of the closing bracket (`]`) is taken as the end point. During conversion a `jump` command is added to all tracks so that they jump to labels at the start and end points. You can also substitute the strings "`loop_start`" and "`loop_end`" in place of the opening and closing brackets, respectively.

3.4 Looping at the Track Level

To loop at the track level, insert Control Change 89 with a value of zero for the loop start point and Control Change 90 (with any value) as the loop end point.

3.5 Empty Space at the Start of an SMF

When SoundMaker converts an SMF, it will automatically remove any empty space up to the first note on command. To prevent this empty region from being cut, add a dummy note having a low volume at the start of the sequence.

Note: If a loop starts at a position before the first note on command, only the empty region before the loop start position will be removed.

3.6 MIDI Events

SMF created using NITRO-Composer for the DS or NW4R SoundMaker for the Wii can be used as is, including, for example, Control Change numbers.

For MIDI events supported by NW4C SoundMaker, see section 8.2 Table of Supported MIDI Control Codes.

3.7 Embedding Text Commands in an SMF

3.7.1 Features

Creating sequence data using the sequencer is useful when you want to quickly check a sound. However, it takes more than MIDI events alone to utilize all of the sequence commands. SoundPlayer therefore includes a feature where text commands can be embedded in an SMF so that they can be output together with MIDI events as sequence commands. Any text command can be output using this feature.

For details on sequence commands, see section4 Text Sequences..

3.7.2 Embedding Text Commands

The following character strings can be embedded in SMF data as markers or text events.

Code 4 Embedding Text Command in SMF

```
text_03:   pan_r 30, 90
```

The specification above will output the command "`pan_r 30, 90`" at the specified location in the third track (MIDI channel).

3.7.3 Outputting All Tracks

Output for all tracks is possible by adding a line like the following.

Code 5 Embedding Text Command to All Tracks of SMF

```
text_all:   pan_r 30, 90
```

Note: Nothing is output for tracks not being used.

3.7.4 Expanding Label Names

An error will result if, for example, you want to define labels at the same position on all tracks using `text_all` as given below.

Code 6 Expanding Label Name to SMF (Error Example)

```
text_all:BLOCK_A:
```

This results in defining the label `BLOCK_A` in multiple locations and results in a multiple label definition error. To achieve the desired effect, use the following instead:

Code 7 Expanding Label Name to SMF

```
text_all:$BLOCK_A:
```

Adding the $ character to the beginning will expand the label names, and data will be output on each track as follows.

Code 8 Example of Expanding a Text Sequence Label Name

```
SMF_filename_Track_0_BLOCK_A:
```

The SMF file name will be inserted at the location *filename* and the track number at the location of the number. A multiple label definition error does not result in this case because the track number differs for each track.

3.7.5 Notes on Real-Time MIDI Playback

Embedded text sequences are not processed during real-time MIDI playback when using SoundMaker PC emulation or a console with MIDI input. Be aware that the result of such real-time MIDI playback will be different from the result of the data converted and played back.

4 Text Sequences

4.1 Overview

It is possible to list more than one set of sequence data in a single file by editing text sequence files (extension `.cseq`) directly with a text editor.

The following is a sample of a text sequence file.

Code 9 Example of Text Sequence

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;   Sample SE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

yoshi:
   prg 0
   cn4 127, 0
   fin

wihaho:
   prg 1
   cn4 127, 0
   fin

; Note command only
; Coin sound
note_only:
   prg 2
   as5 127, 6
   ds6 127, 48
   fin

; Loop by jumping (repeats endlessly)
; Ambulance
jump_seq:
   prg 3
_loop_start:
   bn4 127, 48
   gn4 127, 48
   jump _loop_start
```

4.1.1 Coding Sequence Data

Each sequence starts with a label name definition used to identify the sequence. This label is used as the playback starting position specified by SoundMaker.

A single set of sequence data is created by adding a series of sequence commands after this label definition.

4.2 Numeric Value Notation

For points that specify numeric parameter values in a text sequence file, the following methods can be used in addition to direct numerical input.

4.2.1 Binary and Hexadecimal Values Notation

Although numeric values are usually coded in decimal, you can code values in binary and hexadecimal.

When coding values in binary or hexadecimal, prepend 0b or 0x to the beginning of the value. For example, the decimal value 12 can also be expressed as follows.

Code 10 Describing Numbers in Binary and Hexadecimal

```
0b1100
0xc
```

4.2.2 Bit Notation

Bit notation is useful when coding a value where a given bit can be either 1 or 0, as with bit flags.

Bit notation is coded by specifying which low-order bits are to be set to 1. For example, use the following specification to represent a value where the lower order bits 1, 3, and 6 through 8 have a value of 1.

Code 11 Describing Numbers in Bit Notation

```
{ 1,3,6–8 }
```

This is equivalent to 0b111001010. Note that the lowest order bit is bit 0.

4.2.3 Numerical Formulas

Numerical values can also be represented using numerical formulas. Binary, hexadecimal, or bit notation can be used for each part of a numeric formula.

For example, each of the following represents a valid numerical formula.

Code 12 Describing Numbers as Numerical Formulas

```
2 * 4 + 0x10
( 1 << 4 ) + 3
```

```
{ 0, 2 } | { 4–6 }
```

The table lists the operators that can be used in numeric formulas and their priority.

Table 1 Operators Used to Describe Numbers as Formulas

Priority	Operator	Description
1	*	Multiplication
	/	Division
2	+	Addition
	-	Subtraction
3	>>	Right-shift
	<<	Left-shift
4	<	Left-hand side less than right-hand side
	<=	Left-hand side less than or equal to right-hand side
	>	Left-hand side greater than right-hand side
	>=	Left-hand side greater than or equal to right-hand side
5	==	Left-hand side and right-hand side are equal
6	&	Bitwise AND
7	|	Bitwise OR

5 Sequence Commands

The following table lists the sequence commands that can be used in a text sequence.

Table 2 List of Sequence Commands

Command Name	Description	Default Value
cn4 velocity, length	Note command	
wait length	Rest command	
fin	End sequence	
prg *x*	Change program	0
tempo *x*	Change tempo	120
Timebase *x*	Change time base	48
volume *x*	Change track volume	127
volume2 *x*	Change track volume	127
main_volume *x*	Change player volume	127
pitchbend *x*	Change pitch bend	0
bendrange *x*	Change pitch bend range	2
transpose *x*	Transpose	0
velocity_range *x*	Change velocity range	127
pan *x*	Change track pan	64
span *x*	Change track surround pan	0
init_pan *x*	Change track initial pan	64
prio *x*	Change track voicing priority	64
tieon tieoff	Tie mode start and stop	off
monophonic_on monophonic_off	Monophonic mode on and off	off
notewait_on notewait_off	Note wait on and off	on
porta *x*	Set the portamento start key and start	cn4(60)
porta_time *x*	Set the portamento time	0
damper_on damper_off	Damper pedal on/off	off
porta_on	Start and stop portamento	off

Command Name	Description	Default Value
porta_off		
sweep_pitch *x*	Set the amount of change in the sweep pitch	0
lpf_cutoff *x*	LPF cutoff frequency	0
biquad_type	Biquad filter type	0
biquad_value	Biquad filter value	0
fxsend_a *x*	Effect send A	0
fxsend_b *x*	Effect send B	0
mainsend *x*	Main send	127
alloctrack *mask*	Allocates tracks	
opentrack *no, label*	Start track	
jump *label*	Sequence jump	
call *label*	Call sequence	
ret	Return sequence	
loop_start *count*	Loop start point	
loop_end	Loop end point	
env_hold *x*	Set the hold value	
attack *x*	Set the attack value	
decay *x*	Set the decay value	
sustain *x*	Set the sustain value	
release *x*	Set the release value	
env_adsr *a,d,s,r*	Set the envelope values	
env_ahdsr *a,h,d,s,r*	Set the envelope values (with hold)	
envelope *a,d,s,r*	Set the envelope values (not recommended)	
env_reset	Invalidate envelope values	
mod_depth *x*	Set the modulation depth	0
mod_range *x*	Set the modulation range	1
mod_speed *x*	Set the modulation speed	16
mod_delay *x*	Set the modulation decay	0
mod_type *x*	Set the modulation type	0

Command Name	Description	Default Value
mute *mode*	Track mute / Mute cancellation	0
userproc *procId*	Call a user process	
bank_select *bankId*	Select bank	0

5.1 Note Commands

A note command vocalizes the note corresponding to the current Program Number.

Code 13 Format of Note Command

```
cn4 velocity, length
```

Keys are written in a format similar to the following:

```
cn4, cs4, dn4, ds4, en4, fn4, fs4, gn4, gs4, as4, as4, bn4, cn5
```

`cn4` is used for middle C. Notes can be specified from `cnm1` to `gn9`. (`m1` represents minus 1.)

The `velocity` parameter represents the velocity. Having a value in the range 0 to 127, this is interpreted as a square scale value. In other words, taking 127 as 100% results in the following:

$$\left(Velocity \diagup 127\right)^2 \times 100\%$$

The length of the note is represented by `length`. It can take a value between 0 and 268435455, with 48 representing a quarter note. (The length of the quarter note can be changed using the `timebase` command.) The actual length of a sound is determined by the tempo, so the only value of length that has special meaning is 0.

When `length` is set to 0, the note plays until the end of the waveform data being played. The 0 setting is mainly utilized with looped waveform data. When this is done, the note will keep playing indefinitely until either the program stops the sound or the sound is stopped by something of higher priority.

In the `notewait_on` state (the default state), the sequence process will stop at the place of the note with the sequence process' end and wait for the same length of time as the note length before passing to the next sequence process.

The behavior is the same when `length` = 0, meaning the sequence will remain stopped until the waveform data being played is over. Be aware that even if the waveform data is looped so the note plays indefinitely, the process will move to the next sequence if the voicing priority is low and the sound is forced to stop while the note is playing.

Using this same behavior and setting `length` to 0 for a non-looped set of waveform data, you can specify playback so it passes to the next sequence as soon as the waveform data is over.

By stopping the sequence process for the duration of the `length` value, you are in effect using the

end of the note process to cancel the program's stopped state and resume the sequence.

Code 14 Example of Note Commands

```
cn4 110, 48
dn4 110, 48
en4 127, 96
```

5.2 Wait Command

The `wait` command stops a sequence for the specified length of time.

Code 15 Format of `wait` Command

```
wait length
```

The `length` parameter represents the length of a note, 48 being equivalent to the length of a quarter note. (The length of the quarter note can be changed using the `timebase` command.) Actual length depends on the tempo. This value can be specified in the range 0 to 268435455.

5.3 Finish Sequences

The `fin` command ends sequence processing for the track.

Code 16 Format of the Finish Sequence Command

```
fin
```

When sequence processing for all tracks ends, Player processing also stops.

If you forget to use the `fin` command, sequence data lower in the file will continue to be executed. Be careful as this may result in the output of unexpected sounds or runaway processing in some cases.

5.4 Program Change

The change program command changes the program.

Code 17 Format of the Program Change Command

```
prg x
```

The Program No. value can be specified in the range 0 to 32767. However, only a value up to 127 can be specified under MIDI. The default value is 0.

5.5 Tempo Change

The change tempo command changes the tempo.

Code 18 Format of the Change Tempo Command

```
tempo x
```

Tempo can have a value from 1 to 1023. This value is used in conjunction with a tempo ratio value that can be set by the program to determine the ultimate tempo value. The default value is 120.

5.6 Time Base

This command changes the time base (quarter note resolution).

Code 19 Format of Time Base Command

```
timebase x
```

The time base ranges from 1 to 255. The default value is 48.

5.7 Track Volume

This command changes the volume of a track.

Code 20 Format of Track Volume Command

```
volume x
volume2 x
```

The volume value can be specified in the range 0 to 127. The default value is 127.

Just as with velocity, this value is interpreted as a square scale value. For example, taking 127 as 100% results in the following:

$$\left(Velocity \diagup 127\right)^2 \times 100\%$$

There is no functional difference when using either the `volume` or `volume2` command. If both are specified at the same time, the effect of both will be simultaneously applied. For example, if 80% is specified to volume, and 50% is specified to `volume2`, a volume of 40% (0.8 x 0.5 = 0.4) will result.

5.8 Main Volume

This command changes the volume of the entire sequence.

Code 21 Format of Main Volume Command

```
main_volume x
```

The volume value can be specified in the range 0 to 127. The default is 127.

Just as with velocity, this value is interpreted as a square scale value. For example, taking 127 as 100% results in the following:

$$\left(Velocity \diagup 127\right)^2 \times 100\%$$

5.9 Pitch Bend

This command changes the pitch bend and/or pitch bend range.

Code 22 Format of Pitch Bend/Pitch Blend Commands

```
pitchbend x
bendrange x
```

The `pitchbend` value can be specified in the range –128 to 127. The default value is 0.

The bend range is measured in terms of semitones. The default value is 2. The specifiable range is 0 to 127.

If `pitchbend` is set to MAX, it is changed to the exact height specified by `bendrange`. For example, if `bendrange` is 2, the `pitchbend` value will change +2 half notes at 127, +1 half note at 64, and -1 half note at –64.

5.10 Transpose

This command is used to transpose data.

Code 23 Format of Tranpose Command

```
transpose x
```

The transpose value is measured in semitones and can be set in the range –64 to +63. The default value is 0.

5.11 Velocity Range

This command changes the velocity range of the track.

Code 24 Format of Velocity Range Command

```
velocity_range x
```

The range of values for the velocity range is from 0 to 127. The default value is 127.

When the velocity range value is changed, the note command's velocity value can be changed dynamically. The final velocity value is obtained by multiplying the note command's velocity value by the result of the velocity range value divided by 127.

Although both the `volume` command and the `velocity_range` command ultimately change the volume, the `velocity_range` command differs in that it may change the velocity region.

5.12 Track Pan

This command changes the pan value for the track.

Code 25 Format of Pan Command

```
pan x
```

The pan value can be specified in the range 0 to 127, where a value of 0 represents the left, 127 represents the right, and 64 represents the center. The default value is 64.

5.13 Track Surround Pan

This command changes the surround pan for a track.

Code 26 Format of Track Surround Pan Command

```
span x
```

The surround pan value can be specified in the range 0 to 127, where a value of 0 represents front output only and 127 represents rear output only. The default is 0.

5.14 Track Initial Pan

Changes the track's initial pan.

Code 27 Format of the Track Initial Pan Command

```
init_pan x
```

Unlike normal pan, initial pan can be set for individual notes.

For example, if you set the note wait mode (described in section 5.18 Note Wait Mode) to `notewait off` and play a sequence such as the following, the first cn4 sound is also affected by pan 127, and it changes its panning partway through.

Code 28 Example Using Normal Pan

```
notewait off
pan 64
cn4 127, 96
wait 48
pan 127
en4 127, 96
```

In cases like the following, if you don't want to change the panning of the cn4 that is played first, but you want to change the panning of the en4 that is played after it, initial pan can be used.

Code 29 Using Initial Pan, Example 1

```
notewait off
init_pan 64
cn4 127, 96
wait 48
init_pan 127
```

```
en4 127, 96
```

The following example creates harmony with separate pan settings for each note within a single track, by using note-specific pan settings.

Code 30 Using Initial Pan, Example 2

```
notewait off
init_pan 0
cn4 127, 96
init_pan 64
en4 127, 96
init_pan 127
gn4 127, 96
```

5.15 Voicing Priority

This command sets the voicing priority of a track.

Code 31 Format of Voicing Priority Command

```
prio x
```

The priority value can be specified in the range 0 to 127. The default value is 64.

In reality, the priority value for the player is combined with this priority, resulting in a final voicing priority value. The higher the value, the higher the priority. If two sounds have the same priority value, the most recent sound attempting to play is given priority.

5.16 Tie Mode

This command toggles the tie mode.

Code 32 Format of Tie Mode Command

```
tieon
tieoff
```

The default is tie mode off.

If note on occurs when tie mode is on, the sound will be played continuously regardless of the note length specification made with the note command. If note on is performed again, audio output continues with changes only to the pitch and velocity. Sound output ends when tie mode is turned off or the sequence is stopped.

Turning tie mode on forcibly releases the sound being output. Also, only simple notes are played, even if note wait mode is turned off.

5.17 Monophonic Mode

This command switches monophonic mode.

Code 33 Format of Monophonic mode Command

```
monophonic_on
monophonic_off
```

The default is monophonic mode off.

When monophonic mode is on, sound is limited to one note per track. When the note on command is used when a note is already being voiced for that track, only the pitch and velocity are changed while the sound already being produced continues as-is. Normal note on operations are performed when there is no note being voiced on the track.

If the timing of note off for the immediately previous note is the same as the note on timing of the new note to be produced, the new note to be produced is processed according to normal note on operations because note off operations will be processed first. It is therefore necessary to make notes overlap by at least one tick in order to create a single, continuous note whose pitch and velocity changes.

It is necessary to note that fractional tick values may be rounded when using sequence data converted from SMF. If the resolution after conversion is set to 48, one tick is equivalent to three consecutive 64th notes, and sounds may not be processed as a single, continuous note unless notes of at least three consecutive 64th notes in length are overlapped.

5.18 Note Wait Mode

This command toggles note wait mode.

Code 34 Format of Note Wait Mode Command

```
notewait_on
notewait_off
```

By default, note wait mode is on.

If note on is performed when note wait mode is on, the sequence stops only for the same length as given by the note length specified with the note command. When note wait mode is off, the next command is processed without stopping.

Since the sequence stops while the sound is playing when this mode is turned on, this command is useful for playing notes in order without the constant need to use wait commands. Also, no processing is allowed during sound output.

On the other hand, since there is no need to wait after a note command when this mode is off, it is possible to do things like play two or more notes at the same time or move the pan during sound output.

5.19 Damper Pedal

Performs damper pedal operations.

Code 35 Format of Damper Pedal Command

```
damper_on
damper_off
```

With `damper_on`, the damper (sustain) pedal will enter the pressed state. At this time, the note will be held without being released, even if a `note off` comes.

The held note will be released at the timing of `damper_off`.

5.20 Portamento

This command sets the portamento.

Code 36 Format of Portamento Command

```
porta key
porta_time time
porta_on
porta_off
```

The `porta` command is used to enter portamento mode. If a note command is executed in portamento mode, the pitch specified by key changes to the pitch specified by the note command during playback. When the next note command is executed, the pitch used with the previous note command changes to that specified by the current note command.

The `porta_time` command specifies the speed at which pitch changes. You can specify a value from 0 to 255 (default: 0). The larger the value specified, the slower the change in pitch and the longer it takes. Specifying 0 means that the pitch changes over the length of a note. In other words, pitch changes over the same amount of time as the sound is played by the note command.

Note: In accordance with MIDI conventions, the command used for this is `porta_time`, not `porta_speed`.

The `porta_on` command can also be used to enter portamento mode just like the porta command, but a start key is not specified. In this case, pitch changes from the pitch specified for the note command executed before the `porta_on` command to the note command played after the `porta_on` command. When the next note command is executed, the pitch changes from that specified for the previous note command to that specified for the current note command.

The `porta_off` command releases portamento mode.

You can achieve an effect where the pitch is continuously changing with pitch bend by using portamento during tie mode on.

Note: When this command is used from MIDI while `porta_time` is 0 (default), it will be ignored during real-time MIDI playback.

5.21 Sweep

This command sets the sweep.

Code 37 Format of Sweep Command

```
sweep_pitch pitch
```

Each voice is swept during playback when the `sweep_pitch` command is used. If the note command is executed while the sweep is specified, the sound begins offset by the pitch value specified by the note command, and the pitch is gradually changed to the correct pitch. The pitch value can be specified in the range of –32768 to 32767. The default is 0. Pitch is varied by exactly one half note when 64 is specified.

The speed at which the sound returns to correct pitch is the same as that set using the `porta_time` command.

Note: When this command is used from MIDI while `porta_time` is 0 (default), it will be ignored during real-time MIDI playback.

5.22 LPF Cutoff Frequency

This command changes the cutoff frequency used of the low-pass filter.

Code 38 Format of LPF Cutoff Frequency Command

```
lpf_cutoff x
```

The LPF cutoff frequency can be specified in the range 0 to 127 using 64 as the center. The closer the value approaches 0, the more the filter is applied, and the more the value approaches 127, the more the filter is released. The default value is 64.

Note: The sound manipulation using the current sequence command LPF cutoff frequency will function effectively for only the lower values (64 to 0) where more filters are applied. The LPF cutoff frequency setting is centered at 64 with future access from “instrument” in mind.

5.23 biquad Filter

These commands configure the biquad filter.

Code 39 Format of biquad Filter Commands

```
biquad_type type
biquad_value x
```

When the biquad filter is used, filters such as low-pass filter (LPF), high-pass filter (HPF), or band-pass filter (BPF) can be applied. To change the filter type, set one of the following values with the `biquad_type` command.

Table 3 Biquad Filter Types

type Value	Filter Type
0	Do not use filters. (default value)
1	LPF
2	HPF
3	BPF (Center frequency of 512Hz)
4	BPF (Center frequency of 1024Hz)
5	BPF (Center frequency of 2048Hz)
64	User-defined filter (minimum value)
127	User-defined filter (maximum value)

The `biquad_value` sets how the filter is to be applied. The value ranges from 0 to 127, where 0 means no effect. A value of 127 produces the maximum effect. The default value is 0.

The range of values that can be set by the filter type is from 0 to 127. Of the numbers not currently assigned, be aware that values of 63 or less may have filter types added in the future. However, values of 64 or greater can be used by the user to add custom filter types. Filter types must be added within the program. For details, see the nw::`snd::SoundSystem::SetBiquadFilterCallback` function in the *Revolution Function Reference Manual*. Filter type 64 corresponds to `nw::snd::BiquadFilterType` BIQUAD_FILTER_TYPE_USER_MIN, while filter type 127 corresponds to BIQUAD_FILTER_TYPE_USER_MAX.

5.24 Effect Send A-B

This command changes the amount of data sent to each auxiliary bus. Since these three commands perform the same operation for Auxiliary Bus and B, an example using Effect Send A will be described.

Code 40 Format of Effect Send Command

```
fxsend_a x
```

The effect send level can be specified in the range from 0 to 127, where 0 results in no data being sent and 127 results in the maximum amount of data being sent. The default is 0.

5.25 Main Send

This command changes the volume on the main bus only. Used in combination with the effect send commands for each auxiliary bus, main send only changes the volume of the dry (original sound) component.

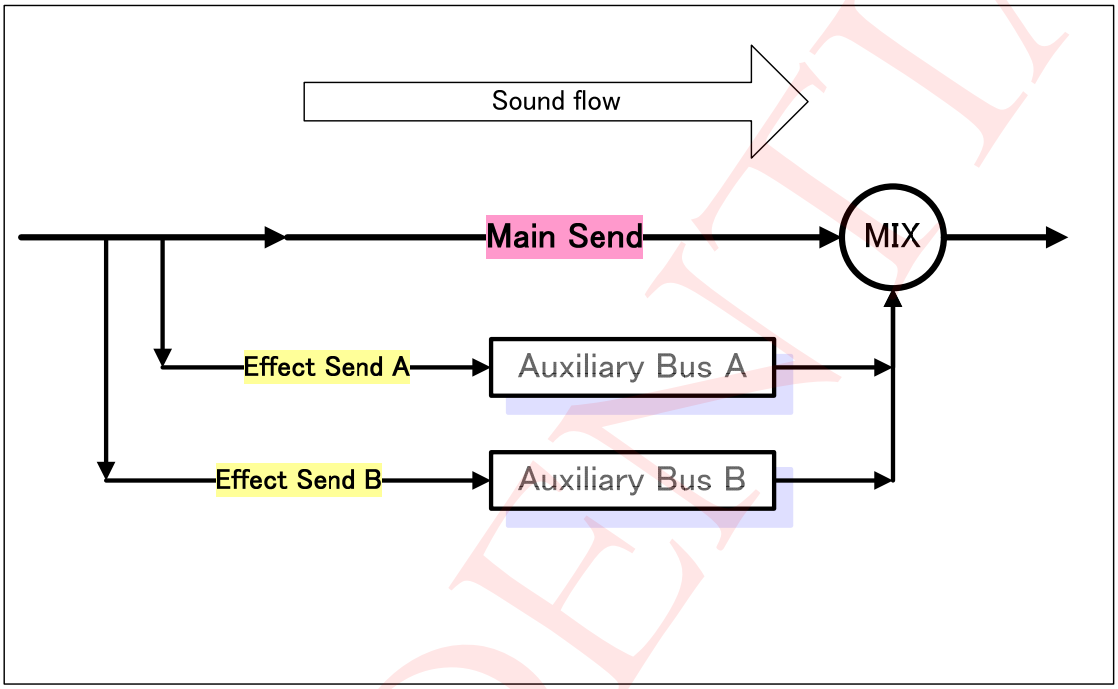
Code 41 Format of Main Send Command

```
mainsend x
```

You can specify the main send volume value from 0 to 127, where 0 represents no dry component, and 127 represents the maximum dry component. The default value is 127.

The following conceptual diagram shows the roles of main send and the two effect send commands during mixing.

Figure 1 Conceptual Diagram of Effect Mixing



A voice can be created for only the wait (effect) component by specifying 0 for Main Send.

5.26 Track Allocation

This command allocates tracks.

Code 42 Format of Track Allocation Command

```
alloctrack mask
```

Note: This command must be used at the very beginning of a sequence.

`mask` is the bit mask of the tracks allocated. Starting from the low order bits, the bit masks represent Track 0, Track 1, Track 2, and so on. (The lowest order bit in the mask is ignored since Track 0 is already allocated.) Using a macro, this command can also be written as

Code 43 Format of Track Allocation Command (Using Macro)

```
allocktrack TRACK_1 | TRACK_2 | TRACK_3
```

Track 1, Track 2, and Track 3 are allocated by the line of code given above.

5.27 Track Start

This command starts another track.

Code 44 Format of Track Start Command

```
opentrack no, label
```

Note: This command can only be executed on tracks that have already been allocated using `alloctrack`.

Although track numbers 0 through 15 can be specified for `no`, the current track cannot be specified. In addition, this command will execute only after all currently executed sounds for tracks are released.

The parameter `label` indicates the start of the sequence.

Tracks terminated by a `fin` command are released, so they cannot be restarted with the `opentrack` command. If you need to restart a track, make the track wait in a loop that contains a `wait` command.

Note: If a track having a Track Number greater than the current track is opened, initial processing will be carried out during the same frame, but if a track having a Track Number lower than the current track number is opened, that processing will be delayed by one frame.

5.28 Sequence Jump

This command switches the sequences position to another location.

Code 45 Format of Sequence Jump Command

```
jump label
```

The `label` parameter represents the location to which to jump.

The `jump` command can be used to create a sequence to be looped. However, an endless loop will result unless a command that stops the sequence, such as the wait command or note wait on command, is included in the loop.

5.29 Sequence Call

Although this command causes the sequence position to jump to another location, it will return to its original position later.

Code 46 Format of Sequence Call Command

```
call label
ret
```

The `label` parameter represents the position being jumped to.

The `ret` command is used at the jump target to return to the position where the call was originally made. Although it is possible to execute the `call` command again from the jump target, such nesting is only allowed up to three levels deep, including the `call` command.

5.30 Loop

This command repeatedly executes a given segment of code the number of times specified.

Code 47 Format of Loop Command

```
loop_start count
loop_end
```

The loop start point is set using `loop_start`. The loop count is specified by `count` in the range 0 to 255. An infinite loop results if 0 is specified.

The loop end point is set using `loop_end`. The process returns to the loop start point specified by `loop_start` until it reaches the number of loops specified by count.

Although looping is allowed inside a loop segment, such nesting is only allowed up to three levels deep, including the `call` command.

If this command is used, it will be ignored during real-time MIDI playback.

5.31 Envelope

This command sets the envelope.

Code 48 Format of Envelope Command

```
attack x
env_hold x
decay x
sustain x
release x
env_adsr a, d, s, r
env_ahdsr a, h, d, s, r
envelope a, d, s, r (not recommended)
```

Although the envelope value of the bank is used by default, the envelope value can be overwritten using this command. If a command such as `release` is executed alone, values set using the bank will still be used for the other values such as `attack`. Multiple values can be set at the same time by using the `env_adsr` or `env_ahdsr` command. The `envelope` command has been maintained for compatibility reasons; use the `env_adsr` command instead.

The hold time value is converted to milliseconds with the following formula and then applied. The maximum value (127) specifies roughly 4 seconds of hold time.

$$\frac{(env\_hold+1)^2}{4}$$

When –1 is specified for each of the envelope values, all changed values can be invalidated. The bank envelope values, which are the default, are used once more for all the invalidated values.

5.32 Envelope Invalidation

This command invalidates the set envelope values.

Code 49 Format of Envelope Invalidation Command

```
env_reset
```

Envelope values set by any envelope command are invalidated, and the envelope values set by the bank are restored. Although it is the same as setting -1 with the `envelope` command, this command invalidates all envelope parameters.

5.33 Modulation

This command sets the modulation.

Code 50 Format of Modulation Command

```
mod_depth depth
mod_range range
mod_speed speed
mod_delay delay
mod_type type
```

Modulation depth is set using `mod_depth`. The specifiable range is 0 to 127 with a default value of 0. The following table gives the degree of variation at maximum values.

Note: This assumes `mod_range`, to be described later, has a value of 1.

Table 5-4 Degree of Variation in Modulation

Type	Degree of Variation
Pitch change	Varies in the range of ±1 one semitone.
Volume change	Varies in the range of ±6.0 dB.
Position change	Varies in the range of left MAX to right MAX.

The maximum degree of modulation is set using `mod_range`. The specifiable range is 0 to 127 with a default value of 1. For example, when varying pitch with a maximum `mod_depth`, pitch can only be varied ±1 half-note. However, it can be modulated over ±12 semitones, or over an entire octave, if `mod_range` is set to 12.

The modulation speed is set using `mod_speed`. The specifiable range is 0 to 127 with a default value of 16. Since modulation speed varies linearly by approximately 0.4 Hz per step, a modulation speed from 0.0 Hz to approximately 50 Hz can be set.

Modulation delay is set using `mod_delay`. The specifiable range is 0 to 32767 with a default value of 0. Measured in units of one sound frame (approx. 3 ms), this value is independent of the tempo. This parameter is set from MIDI using Control Change 26 or 27. Although the value is output as is when

Control Change 26 is used, while a value of ten times is output when Control Change 27 is used. As a result, modulation delay can effectively be set in the range 0 to 1270.

The modulation type is set using `mod_type`. Specifiable values are as given in section 8.1 List of All Sequence Commands. The default is 0 = vibrato (varies pitch).

You can use a number to specify the modulation type, or you can specify it using one of the listed macro character strings.

Table 5 List of Modulation Types

Value	Macro	Description
0	MOD_TYPE_PITCH	Vibrate (pitch change)
1	MOD_TYPE_VOLUME	Tremolo (volume change)
2	MOD_TYPE_PAN	Pan (location change)

Note: Amplitudes do not change beyond the threshold. For example, if the original volume is already at maximum and an attempt is made to vary that volume, it will not become louder; instead, it will only become softer.

5.34 Track Mute

This command controls the muting of tracks.

Code 51 Format of Track Mute Command

```
mute mode
```

Various mute operations are performed depending on the numeric value of `mode`.

Table 6 List of Track Mute Operations

Value	Description
0	Release track mute
1	Mutes the track, and sounds already playing are allowed to continue
2	Mutes the track, and sounds already playing are released and stopped
3	Mutes the track, and sounds already playing are immediately stopped

5.35 Calling a User Process

This command calls a user process registered with a program.

Code 52 Format of User Process Call Command

```
userproc procId
```

When writing the `userproc` command, a function already registered with the program is called at the time the command is encountered. The numeric value of `procId` is passed to the function as a

parameter.

The following actions are possible inside the function registered as a user process.

- Access and change sequence variables
- Access and change the results of a comparison command (see section below).

Furthermore, since program code can be written freely within the user process, it is possible to perform complex calculations not possible when using just sequence commands or execute processes dependent on game scenes and parameters, and then take the result and apply it to sequence data by storing it in a sequence variable or in a comparison command result.

5.36 Select Bank

Sets the track's bank number.

Code 53 Format of Select Bank Command

```
bank_select bankIndex
```

SoundMaker can register up to four banks for a single sound. This command lets you set which bank to use.

`bankIndex` can specify numbers 0 to 3, which correspond to the line numbers of the list shown in SoundMaker's **Multibank Settings** dialog.

5.37 Front Bypass

This command specifies whether to turn front bypass On or Off.

Code 54 Format of Front Bypass Command

```
frontbypass_on
frontbypass_off
```

The default is the value specified on the sequence list of SoundMaker. By using the appropriate command you can overwrite the value specified by SoundMaker, except in the case of the track being used.

6 Extended Sequence Commands

The extended sequence commands are a group of commands that allow settings to be made with a greater degree of freedom than the sequence commands described up to this point.

Although using the usual sequence commands is enough when playing back conventional sequences, use of extended sequence commands allows more elaborate sequences and interactive sequence playback among other enhancements.

6.1 Time Change Commands

It is possible to specify a time change for several of the sequence commands. Parameters are changed smoothly for the specified amount of time.

```
volume_t 64, 48
```

In the example above, the volume level is changed from the current value to a value of 64 for 48 ticks.

The same parameter value as the conventional command is manipulated when using a time change command. In other words, if 0 is specified for the time with a time change command, the result is the same as using the conventional command.

In addition, when using time change commands, the last command specified is always enabled. For example, if a new time change volume command is executed while varying the volume with another time change command, the new time change will start from the current value.

6.1.1 Format

The format of a time change command is as follows.

```
(command)_t x, time
```

`x` is a command parameter, and `time` is the change time. Tick units are used for the time unit just as with the note command and wait command. The range of values that can be specified for time is 0 though 32767.

6.1.2 List of Time Change Commands

Table 7 List of Time Change Commands

`volume_t`	`pan_t`	`span_t`	`pitchbend t`

6.1.3 Extensions to Time Change Commands

A random or variable value can be used for the specified time value. For details, see section 6.5 Combining Extended Sequence Commands.

6.2 Random Commands

A random number can be set for the value for several sequence commands.

```
pitchbend_r –12 , 12
```

In the example above, a random value from –12 to +12 is used when the `pitchbend` command is executed. This allows the pitch to be changed slightly each time the sound is produced.

6.2.1 Format

The basic format of the random command is shown below.

```
(command)_r min, max
```

A random number in the range between `min` and `max` is generated, and the command is executed using that value as its parameter.

6.2.2 List of Random Commands

Table 8 List of Random Commands

wait_r	prg_r	tempo_r	volume_r
volume2_r	main_volume_r	pitchbend_r	pan_r
transpose_r	porta_time_r	sweep_pitch_r	mod_depth_r
mode_speed_r	attack_r	decay_r	sustain_r
release_r	mod_delay_r	loop_start_r	mute_r
bank_select_r			
setvar_r	addvar_r	subvar_r	mulvar_r
divvar_r	shiftvar_r	randvar_r	randvar_r
orvar_r	xorvar_r	notvar_r	modvar_r
cmp_eq_r	cmp_ge_r	cmp_gt_r	cmp_le_r
cmp_lt_r	cmp_ne_r		

For details on `setvar_r` and subsequent commands in the table, see descriptions in section 6.3 Variable Commands and section 6.4 Conditional Commands.

For a list of supported sequence commands, see section 8.1 List of All Sequence Commands.

6.2.3 Random Note Length Command

A note command can be made to act like a random command by appending `_r` to the end. This command sets the note length to a random range of values.

```
cn4_r velocity, min, max
```

To change the pitch of a sound randomly, use `transpose_r`. To change the velocity of a sound randomly, use `volume_r`, `volume2_r`, and so on.

6.3 Variable Commands

Variables can be used within a sequence.

6.3.1 What Is a Variable?

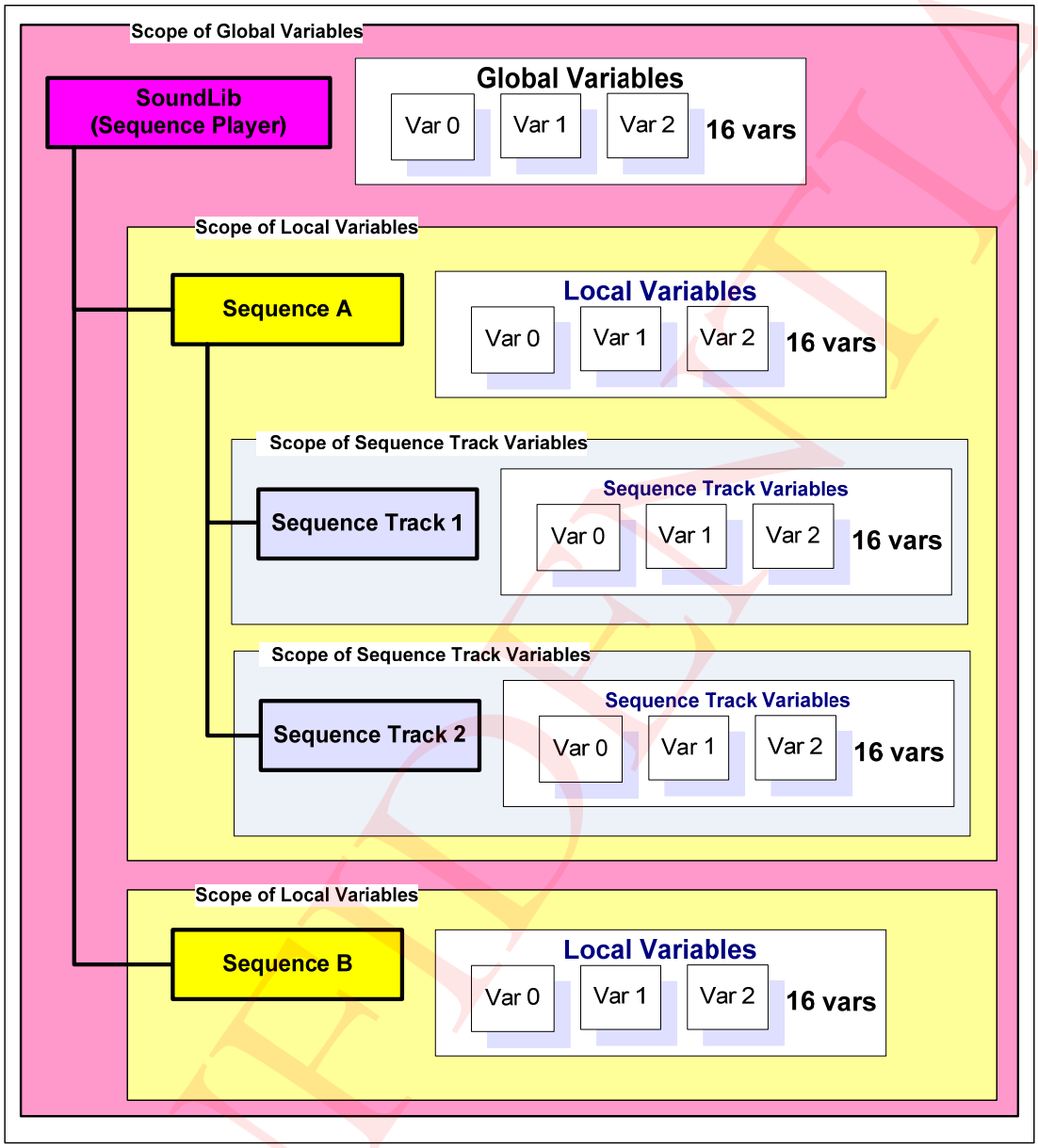
Variables are locations in memory that allow the storage of numeric values.

Any value in the range of –32768 to 32767 can be stored in a single variable.

Each sequence can make free use of 16 variables that are exclusive to that sequence for each sequence track.

Variables used within sequences are called local variables. Variables that are shared and can be freely used by all sequences are called global variables. Variables that only the sequence track can make free use of are called sequence track variables.

Figure 2 Local Variables and Global Variables



Each variable inside a sequence is specified by a number.

The numbers 0 to 15 indicate one of the 16 local variables.

The numbers 16 to 31 indicate one of the 16 global variables. The actual variable number is the value that results from subtracting 16 from this number. Thus, numbers 16 to 31 actually represent the global variables 0 to 15.

Similarly, numbers 32 to 47 indicate the 16 sequence track variables. Because 32 is subtracted from each number, they actually represent the sequence track variable 0 to 15.

The default value for the variables is –1. Global variables are initialized when the system starts. Local variables and sequence track variables are initialized when the sequence starts.

6.3.2 Specifying Variables Using Character Strings

Variable numbers can also be specified using macro character strings. You can use these macros instead of `varNo` to specify variable numbers in the variable commands explained below.

Table 6-3 is a list of the macros for specifying variable numbers.

Table 9 Macros Specifying Variable Numbers

Macro	Description
VAR_0, VAR_1 … VAR_15	Corresponds to local variables 0 to 15
GVAR_0, GVAR_1 … GVAR_15	Corresponds to global variables 0 to 15
TVAR_0, TVAR_1 … TVAR_15	Corresponds to sequence track variables 0 to 15

6.3.3 Variable Calculation Commands

Values can be set and four arithmetic operators and other operators can be used for each variable.

Table 10 List of Variable Operation Commands

Command Name	Description
`setvar` *varNo, x*	Sets the variable to value *x*
`addvar` *varNo, x*	Adds *x* to the value of the variable
`subvar` *varNo, x*	Subtracts *x* from the value of the variable
`mulvar` *varNo, x*	Multiplies the value of the variable by *x*
`divvar` *varNo, x*	Divides the value of the variable by *x*
`shiftvar` *varNo, x*	Left shifts the variable value by *x* bits (right shifts if a negative value is used)
`randvar` *varNo, x*	Sets a random value between 0 and *x* in the variable (or in the range of *x* to 0 if the value is negative)
`andvar` *varNo, x*	Logical product of variable's value and each of x bits
`orvar` *varNo, x*	Logical sum of variable's value and each of x bits
`xorvar` *varNo, x*	Exclusive sum of variable's value and each of x bits
`notvar` *varNo, x*	Negation of variable's value and each of x bits
`modvar` *varNo, x*	Remainder calculation for variable's value and x

6.3.4 Variable Commands

Sequence commands can be executed using the variable set with the variable calculation commands described above.

```
pitchbend_v 0
```

In the preceding example, the `pitchbend` command is executed using the local variable `0`.

6.3.5 Format

The basic format of a variable command is as follows.

```
(command)_v varNo
```

Here, the command is executed using the variable value specified by `varNo`.

Note: A value range was set for the arguments for each sequence command, and operations cannot be guaranteed when a variable is set to a value that is outside this range.

6.3.6 List of Variable Commands

All sequence commands that can be used as random commands may also be used as variable commands.

Table 11 List of Variable Commands

wait_v	prg_v	tempo_v	volume_v
volume2_v	main_volume_v	pitchbend_v	pan_v
transpose_v	porta_time_v	sweep_pitch_v	mod_depth_v
mode_speed_v	attack_v	decay_v	sustain_v
release_v	mod_delay_v	loop_start_v	mute_v
setvar_v	addvar_v	subvar_v	mulvar_v
divvar_v	shiftvar_v	randvar_v	andvar_v
orvar_v	xorvar_v	notvar_v	modvar_v
bank_select_v			
cmp_eq_v	cmp_ge_v	cmp_gt_v	cmp_le_v
cmp_lt_v	cmp_ne_v		

Commands that start with `cmp_` are comparison commands and will be described later in section 6.4 Conditional Commands.

For a table showing compatibility with regular sequence commands, see section 8.1 List of All Sequence Commands.

6.3.7 Note Length Variable Note Command

A note command can be made into a variable command by attaching `_v`. This allows the note length specification to be made using a variable.

```
cn4_vvelocity varNo
```

To specify the pitch using a variable, use `transpose_v`. To specify the velocity using a variable use `volume_v` or `volume2_v`, and so on.

6.3.8 Calculation Between Variables Commands

The variable calculation commands described above can be used to add or subtract a value to or from a variable. Variable commands such as `setvar_v` can be used for variable substitution and when adding the value of one variable to another variable.

Code 55 Example of Operations Between Variables

```
setvar_v 0, 1  ; Assign the value of Variable 1 to Variable 0
addvar_v 2, 3  ; Add the value of Variable 2 and Variable 3
```

6.4 Conditional Commands

A variable can be used to control whether a sequence command executes.

6.4.1 Comparison Commands

A comparison command must execute before using conditional commands. The decision whether to execute the conditional command is based on the result of comparing two values using a comparison command.

The following table lists the available comparison commands.

Table 12 List of Comparison Commands

Command Name	Format	Description
cmp_eq *varNo*, *x*	(variable) == x	True if the variable value equals x
cmp_ge *varNo*, *x*	(variable) >= x	True if the variable value is greater than or equal to x
cmp_gt *varNo*, *x*	(variable) > x	True if the variable value is greater than x
cmp_le *varNo*, *x*	(variable) <= x	True if the variable value is less than or equal to x
cmp_lt *varNo*, *x*	(variable) < x	True if the variable value is less than x
cmp_ne *varNo*, *x*	(variable) != x	True if the variable value is not equal to x

If the result of a comparison is true, the following conditional command is executed. If false, the following conditional command is not executed.

6.4.2 Conditional Commands

Conditional commands exist for all sequence commands. Conditional commands use the same command name as the original function with `_if` appended to the end.

```
pitchbend_if +48
```

In the following example, a given sound is turned on and off based on a random number.

Code 56 Example of a Conditional Command

```
randvar 0, 100    ; Sets a random value from 0 to 100
cmp_le 0, 80      ; True if the random value is 80 or lower
cn4 96, 12
dn4_if 96, 12     ; Sound is played only if true
en4 96, 12
fin
```

6.5 Combining Extended Sequence Commands

The extended sequence commands described so far can be combined together for simultaneous use.

6.5.1 Extensions to Time Change Commands

In the case of commands that correspond to time change commands, it is possible to extend the time change command and use a random or variable value for the change time value.

```
(command)_tr x, timeMin, timeMax
(command)_tv x, timeVarNo
```

To use a random value for the change time, specify `_tr` in place of `_t`. To specify a variable value for the change time, specify `_tv` in place of `_t`.

6.5.2 Combining Extended Sequence Commands

Extended sequence commands can be combined according to a set rule. Only extensions supported by a command can be combined.

```
 (command){_r,_v}{_t,_tr,_tv}{_if}
```

Specify `_r` for random commands and `_v` for variable commands first, followed by `_t` for a time change command. Specify `_if` for a conditional command last. This specification order cannot be changed.

For example, you can specify a command such as the following.

```
volume_r_tv_if min, max, timeVarNo
```

This command sets a random volume ranging between *min* and *max* for the time given by the variable `timeVarNo` if the result of the comparison command is true.

6.6 Communication with Programs

Communication with programs can be performed using variables. Since variables can be read and written by programs, they can be used for passing information back and forth.

6.6.1 Communication with MIDI Sequences

Information from a MIDI sequence can be notified to a program using variables.

The eight Control Changes 16 through 19 and 80 through 83 are each converted by the `setvar` command. Control Change 16 is set to Local Variable 0. Similarly, Control Changes 17, 18 and 19 are set to Local Variables 1, 2 and 3, respectively. Control Change 80 is set to Track Variable 0, while Control Changes 81, 82, and 83 are set to Track Variables 1, 2, and 3, respectively.

For example, by setting these variables at the break between each block making up a track, it is possible for the program to know which block is currently being played.

6.6.2 Combining with Conditional Commands

It is possible to change sequences at a certain time in a program by combining conditional commands with the reading and writing of variables from the program.

Code 57 Example of Combining with Conditional Commands

```
_loop:
  cmp_eq 0, 1
  jump_if _loop2  ; Jump only when Variable 0 is 1
  cn4 96, 12
  dn4 96, 12
  jump _loop
_loop2:
  cn4 96, 12
  en4 96, 12
  jump _loop2
```

In the above example, the sequence of notes *C*, *D*, *C*, *D*… is played repeatedly as long as the program does nothing. However, this sequence changes to *C*, *E*, *C*, *E*… when a value of 1 is assigned to Local Variable 0 by the program.

6.7 Debugging Variables

Sometimes operations are not performed as expected due to errors that occur when trying to handle complicated variable processing. In such a case, a sequence command is provided to check whether the intended value is assigned to the used variable.

Table 13 List of Variable Debugging Commands

Command Name	Description
printvar *varNo*	Debug output of the value of the variable

Debug output such as shown on the line below is made when this command is processed.

```
#8077a450[1]: printvar GVAR_2(18) = -1
```

The part, `#8077a450[1]`, appearing before the colon indicates that the player identifier is 8077a450 and that the track number is 1. The numeric value of the player identifier itself does not have any important significance. The player identifier is merely used to show that output is from another sequence if the player identifier differs. The track number indicates the track from which the sequence was output.

The data stored by the variable appears following `printvar`. `VAR_` is displayed for local variables, `GVAR_` for global variables, and `TVAR_` for track variables, followed by the variable number. The value inside parentheses is the variable's serial number when used by sequence data. In the example above, the information appearing after `printvar` indicates that the value of global variable 2 is -1.

Output from `printvar` is enabled for SoundPlayer. Because output from `printvar` is disabled by default for user programs, you must enable it using the `SoundSystem::EnableSeqPrintVar` function. For details, see the Function Reference Manual.

7 Preprocessor Directives

7.1 Overview

Preprocessor directives can be used for all text sequences. They support text coding.

For example, one preprocessor directive is called `#define` and is used as follows.

```
#define LENGTH 24
```

By inserting the above line of code in a text sequence, you can write `LENGTH` in place of the number 24 in lines of code that follow. If there are many locations in code that use this value, it can be changed throughout the code simply by editing the value 24 in the `#define` line.

This is extremely useful when selecting a value by trial and error.

7.2 Preprocessor Directives

The following table lists the available preprocessor directives.

Table 14 List of Preprocessor Directives

Preprocessor Directive	Description
#define	Defines substitution macros
#undef	Disables a substitution macro definition
#include	Includes other files
#if	Evaluates a true-false condition
#ifdef	Evaluates whether a macro is defined
#ifndef	Evaluates whether a macro is undefined
#else	Indicates the beginning of segment for items evaluated FALSE in `#if`, `#ifdef`, `#ifndef`, or `#elif`
#endif	Indicates the end of a segment that starts with `#if`, `#ifdef`, `#ifndef`, or `#elif`
#elif	Evaluates conditions for items deemed FALSE in `#if`, `#ifdef`, or `#ifndef`

The short descriptions given in the table may be somewhat hard to understand, but you should be able to understand quickly by reading through the examples that follow.

7.3 #define

This directive defines substitution macros. The syntax is shown below.

```
#define macro_name substitution_string
```

If a character string matching the given macro name appears on any lines that follow, that string will

be replaced by the *substitution* string. Macro names are case-sensitive and are only replaced when they match exactly.

Just as with label names, macro names must start with an alphabetic character, while the rest of the name can consist of alphanumeric characters and underscores (_). Usually, lowercase alphabetic characters are not used.

The replacement string can be specified freely, but, it must match the syntax of the location where the macro is used. For example, a macro used in a location where a numeric value will be written must be defined using a replacement string that represents a numeric value. It is omitted where a macro needs to be defined only for use in `#ifdef`.

Although an error results if the same macro name is defined more than once, you can redefine a macro name if it is first disabled using `#undef`.

7.3.1 Examples of Using #define

```
#define LENGTH 24

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

The above code is the same as that given below.

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

The following code also has the same meaning because the replacement string can be specified freely.

```
#define VEL_LEN 127, 24

   cn4 VEL_LEN
   dn4 VEL_LEN
   en4 VEL_LEN
   fin
```

The following example combines a macro definition with `#ifdef`.

```
#define ENABLE_FLAG

   cn4 127, 24
   dn4 127, 24
#ifdef ENABLE_FLAG
   en4127, 24
#endif
   fin
```

The line between `#ifdef` and `#endif` is executed only if `ENABLE_FLAG` is defined. If the line `#define ENABLE_FLAG` is deleted, the line `en4 127, 24` will be ignored.

7.4 #undef

This directive disables a substitution macro definition. The syntax is shown below.

```
#undef macro_name
```

This directive disables a macro defined using `#define`, causing it to be undefined. The directive does nothing if a macro that has not been defined is specified.

This macro is used to redefine a macro or in combination with `#ifdef/#ifndef`.

7.4.1 Example of Using #undef

```
#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
```

If `#undef` is used in this way, the same macro name can be defined twice.

```
#define ENABLE_FLAG
#undef ENABLE_FLAG

   cn4 127, 24
   dn4 127, 24
#ifdef ENABLE_FLAG
```

```
   en4 127, 24
#endif
   fin
```

The line between `#ifdef` and `#endif` is executed only if `ENABLE_FLAG` is defined. Here, the line `en4 127, 24` is disabled because the macro has been disabled using `#undef`.

7.5 #include

This directive includes another file. The syntax is as follows.

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

The file specified by `filename` is embedded at the current position. Use `#include` to collect items common to several files into one file, or to divide a file when you want several people to edit it.

Both relative and absolute paths can be used for the file name. If the file name is specified using a relative path, the directory used as the base differs depending on which of the two types of available syntax is used. If syntax of the form `#include "filename"` is used, it will be a relative path from the current file. If syntax of the form `#include <filename>` is used, it will be a relative path from the sound project file to be converted.

7.5.1 Example of Using #include

When a bank file is converted, lines of code such as the following are output to a Program Number List File with extension of `*.cinl`.

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

Including this file at the start of a text sequence allows labels to be used with program changes.

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

  prg PRG_ORGAN
  cn4 127, 24
  fin
```

7.6 #if

This directive is used to evaluate true-false conditions. The syntax is as follows.

```
#if evaluation_expression
(enabled when true)
#endif
```

If the result of the evaluation expression is true, the following lines of code up to the next `#endif` are enabled. Otherwise, those same lines of code are disabled.

Note: You can specify a numeric value as the expression to evaluate. When the numeric values are specified, the code is disabled if the value is zero and enabled if the value is non-zero.

7.6.1 Examples of Using #if

Lines of code can be disabled as shown below and used in place of comments.

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

Writing code this way makes it easy to enable these lines later.

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

The following is also an allowed usage.

```
#define VERSION 2

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

This feature allows lines of code to be enabled or disabled based on the evaluation of a given expression. In this example, the line `en4 127, 24` is enabled because the result of evaluation is true.

7.7 #ifdef / #ifndef

This directive evaluates whether a macro is defined or undefined. The syntax is as follows.

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

Specify the macro name instead of an evaluation expression to check whether the macro name is defined. The result of `#ifdef` is true if the macro is defined, while the result of `#ifndef` is true if the macro is undefined. All other operations are identical to `#if`.

7.7.1 Examples of Using #ifdef / #ifndef

```
#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 above example, the line `en4 127, 24` is enabled because `ENABLE_FLAG` is defined, while the line `fn4 127 24` is disabled. Conversely, if `#undef` is inserted as follows:

```
#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
```

The line `fn4 127, 24` is enabled because `ENABLE_FLAG` is undefined.

7.8 #else

`#else` marks the beginning of the code that will execute when the FALSE condition is met for `#if`, `#ifdef`, `#ifndef`, or `#elif`. The basic syntax is as follows.

```
#if ...
(code enabled when true)
#else
(code enabled when false)
#endif
```

The code segment between `#else` and `#endif` is enabled if the evaluation result of the `#if` or `#ifdef` is false. Conversely, the code segment between `#else` and `#endif` is disabled if the

evaluation result is true.

7.8.1 Example of Using #else

Writing code as follows allows two versions of code to be easily switched.

```
#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 code segment between `#if` and `#else` is enabled, but the code segment between `#else` and `#endif` can be enabled instead simply by changing `#if 1` to `if 0`.

7.9 #endif

This directive indicates the end of a code segment beginning with `#if`, `#ifdef`, `#ifndef`, or `#elif`. The basic syntax is as follows.

```
#if evaluation_expression
(code enabled when true)
#endif
```

For details, see the descriptions of `#if`, `#ifdef`, `#ifndef`, and `#elif`.

7.10 #elif

`#elif` evaluates the condition for the FALSE returned from `#if`, `#ifdef`, or `#ifndef`. The basic syntax is as follows.

```
#if evaluation_expression_1
#elif evaluation_expression_2
#endif
```

Please see the following section, as an example may be easier to understand than a verbal description.

7.10.1 Examples of Using #elif

```
#define VERSION 2

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

Code can be made easier to read by rewriting the above using `#elif` as shown below.

```
#define VERSION 2

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

8 Appendix

8.1 List of All Sequence Commands

Commands marked as “variables” can be used as variable commands as well as random commands

Table 15 List of All Sequence Commands

Command	Description	Default value	Range	Time Change	Variable
cn4 *velocity*, *length*	Note command				○
wait *x*	Wait command				○
fin	Finish sequence				
prg *x*	Program change	0	0 – 32767		○
tempo *x*	Tempo change	120	1 – 1023		○
timebase *x*	Time base (quarter note resolution)	48	1 –255		
volume *x*	Track volume	127	0 – 127	○	○
volume2 *x*	Track volume (expression)	127	0 – 127		○
main_volume *x*	Main volume	127	0 – 127		○
pitchbend *x*	Pitch bend	0	-128 – 127	○	○
bendrange *x*	Pitch bend range	2	0 – 127		
transpose *x*	Transpose	0	-64 – 63		○
velocity_range *x*	Velocity range	127	0 – 127		○
pan *x*	Track pan	64	0 – 127	○	○
span *x*	Track surround pan	0	0 – 127	○	○
init_pan *x*	Track initial pan	64	0 – 127		○
prio *x*	Track voicing priority	64	0 – 127		
tieon tieoff	Tie mode on and off	off			
monophonic_on monophonic_off	Monophonic mode on and off	off			
notewait_on notewait_off	Note wait on and off	on			
porta *x*	Portamento start key setting and portamento on	cn4(60)	0 – 127		

Command	Description	Default value	Range	Time Change	Variable
porta_time *x*	Portamento time	0	0 – 255		○
damper_on damper_off	Damper pedal on and off	off			
porta_on porta_off	Portamento on and off	off			
sweep_pitch *x*	Sweep pitch	0	-32768 – 32767		○
biquad_type *type*	Biquad filter type	0	0 – 127		○
biquad_value *x*	Biquad filter value	0	0 – 127		○
fxsend_a *x*	Effect send A	0	0 – 127		○
fxsend_b *x*	Effect send B	0	0 – 127		○
mainsend *x*	Main send	127	0 – 127		○
alloctrack *mask*	Allocate track				
opentrack *no*, *label*	Start track				
jump *label*	Sequence jump				
call *label*	Sequence call				
ret	Sequence return				
loop_start *x*	Loop start		0 – 255		○
loop_end	Loop end				
attack *x*	Envelope attack		-1 – 127		○
env_hold *x*	Envelope hold		-1 – 127		○
decay *x*	Envelope decay		-1 – 127		○
sustain *x*	Envelope sustain		-1 – 127		○
release *x*	Envelope release		-1 – 127		○
env_adsr *a*,*d*,*s*,*r*	Envelope ADSR				
env_ahdsr *a*,*h*,*d*,*s*,*r*	Envelope AHDSR				
envelope *a*,*d*,*s*,*r*	Envelope ADSR (not recommended)				
env_reset	Envelope invalidation				
mod_depth *x*	Modulation depth	0	0 – 127		○

Command	Description	Default value	Range	Time Change	Variable
mod_range *x*	Modulation range	1	0 – 127		
mod_speed *x*	Modulation speed	16	0 – 127		○
mod_delay *x*	Modulation delay	0	0 – 32767		○
mod_type *x*	Modulation type	0	0 – 2		
mute *mode*	Mutes a track/cancels mute	0	0 – 3		○
userproc *procId*	Calls a user process		0 – 65535		○
bank_select *x*	Select bank	0	0 – 3		○
frontbypass_on frontbypass_off	Front bypass				
setvar *no*, *x*	Variable assignment				○
addvar *no*, *x*	Variable addition				○
subvar *no*, *x*	Variable subtraction				○
mulvar *no*, *x*	Variable multiplication				○
divvar *no*, *x*	Variable division				○
shiftvar *no*, *x*	Variable shift				○
randvar *no*, *x*	Random value assignment				○
andvar *no*, *x*	Bit calculation, logical product				○
orvar *no*, *x*	Bit calculation, logical sum				○
xorvar *no*, *x*	Bit calculation, exclusive sum				○
notvar *no*, *x*	Bit calculation, negation				○
modvar *no*, *x*	Remainder calculation				○
printvar *no*	Print debut variable				
cmp_eq *no*, *x*	Comparison (==)				○
cmp_ge *no*, *x*	Comparison (>=)				○
cmp_gt *no*, *x*	Comparison (>)				○
cmp_le *no*, *x*	Comparison (<=)				○
cmp_lt *no*, *x*	Comparison (<)				○
cmp_ne *no*, *x*	Comparison (!=)				○

8.2 Table of Supported MIDI Control Codes

- For #13 transpose, subtracts 64 from the value.
- #16 to #19 set values of the local variables 0 through 3, respectively.
- For #28 and #29, subtract 64 from the value for `sweep_pitch` (for #29, the `sweep_pitch` is then multiplied by 24).
- For #64 `damper_on` / `damper_off`, it is `damper_on` if the value is 64 or greater, and `damper_off` if the value is less than 64.
- For #65, `porta_on` if the value is 64 or higher, and `porta_off` if the value is less than 64.
- For #68, `monophonic_on/monophonic_off,` it is `monophonic_on` when the value is 64 or more, and `monophonic_off` when the value is less than 64.
- For #119, `frontbypass_on / frontbypass_off` it is `frontbypass_on` when the value is 64 or more, and `frontbypass_off` when the value is less than 64.
- #120 to #127 are channel mode messages and not control changes.
- RPMs (“Registered Parameter Number”) #100 and #101 currently only support RPN0 Pitch Bend Sensitivity.
- Data entry supports only #6 ”Data Entry MSB.”

Table 16 Table of Supported MIDI Control Codes

Decimal	Hex	Command		Decimal	Hex	Command
0	0x00	bank_select		64	0x40	damper_on / damper_off
1	0x01	mod_depth		65	0x41	porta_on / porta_off
2	0x02			66	0x42	
3	0x03	init_pan		67	0x43	
4	0x04			68	0x44	monophonic_on / monophonic_off
5	0x05	porta_time		69	0x45	
6	0x06	Data Entry MSB		70	0x46	
7	0x07	volume		71	0x47	
8	0x08			72	0x48	
9	0x09	span		73	0x49	
10	0x0A	pan		74	0x4A	
11	0x0B	volume2		75	0x4B	
12	0x0C	main_volume		76	0x4C	
13	0x0D	transpose		77	0x4D	
14	0x0E	prio		78	0x4E	
15	0x0F			79	0x4F	env_hold

Decimal	Hex	Command		Decimal	Hex	Command
16	0x10	setvar VAR_0		80	0x50	setvar TVAR_0
17	0x11	setvar VAR_1		81	0x51	setvar TVAR_1
18	0x12	setvar VAR_2		82	0x52	setvar TVAR_2
19	0x13	setvar VAR_3		83	0x53	setvar TVAR_3
20	0x14	bendrange		84	0x54	porta
21	0x15	mod_speed		85	0x55	attack
22	0x16	mod_type		86	0x56	decay
23	0x17	mod_range		87	0x57	sustain
24	0x18			88	0x58	release
25	0x19			89	0x59	loop_start
26	0x1A	mod_delay		90	0x5A	loop_end
27	0x1B	mod_delay (x 10)		91	0x5B	fxsend_a
28	0x1C	sweep_pitch		92	0x5C	fxsend_b
29	0x1D	sweep_pitch (x 24)		93	0x5D	
30	0x1E	biquad_type		94	0x5E	
31	0x1F	biquad_value		95	0x5F	mainsend
32	0x20			96	0x60	
33	0x21			97	0x61	
34	0x22			98	0x62	NRPN LSB
35	0x23			99	0x63	NRPN MSB
36	0x24			100	0x64	RPN LSB
37	0x25			101	0x65	RPN MSB
38	0x26			102	0x66	
39	0x27			103	0x67	
40	0x28			104	0x68	
41	0x29			105	0x69	
42	0x2A			106	0x6A	
43	0x2B			107	0x6B	
44	0x2C			108	0x6C	
45	0x2D			109	0x6D	

Decimal	Hex	Command		Decimal	Hex	Command
46	0x2E			110	0x6E	
47	0x2F			111	0x6F	
48	0x30			112	0x70	
49	0x31			113	0x71	
50	0x32			114	0x72	
51	0x33			115	0x73	
52	0x34			116	0x74	
53	0x35			117	0x75	
54	0x36			118	0x76	
55	0x37			119	0x77	frontbypass_on/ frontbypass_off
56	0x38			120	0x78	
57	0x39			121	0x79	
58	0x3A			122	0x7A	
59	0x3B			123	0x7B	
60	0x3C			124	0x7C	
61	0x3D			125	0x7D	
62	0x3E			126	0x7E	
63	0x3F			127	0x7F	

8.3 MIDI NRPN Correspondence Table

Table 17 MIDI NRPN Correspondence Table

MSB (Decimal)	MSB (Hexadecimal)	LSB (Decimal)	LSB (Hexadecimal)	Command
0	0x00	0	0x00	env_reset

9 Revision History

Version	Revision Date	Description
1.5.0	2010/11/04	• Deleted the notes in section 5.22 LPF Cutoff Frequency and section 5.23 biquad Filter which said commands were not enabled, because as of NW4C-1.2.0 they now are enabled. Also made small adjustments to the sections.
1.4.0	2010/08/23	• Added section 5.37 Front Bypass
1.3.0	2010/07/29	• Changed section 5.29 Sequence Call and section 5.30 Loop Mentioned that the three-level-deep nesting restriction includes the `call` command. • Changed section 8.1 List of All Sequence Commands and section 8.2 Table of Supported MIDI Control Codes Added text about `frontbypass_on` and `frontbypass_off`
1.2.0	2010/06/28	• Added text about `bank_select_v` and `bank_select_r` to section 6.2.2 List of Random Commands, section 6.3.6 List of Variable Commands, and section 8.1 List of All Sequence Commands
1.1.0	2009/11/11	• Added note(s) stating that this document is based on NintendoWare for Revolution. • Revised some text to refer to “CTR” instead of “Revolution”.
1.0.0	2009/10/30	• Initial version.