NITRO-Composer
Sequence Data Manual

Version 1.6.0

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

Confidential

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

Table of Contents

Code

Tables

Revision History

Version	Revision Date	Description
1.6.0	03/28/2005	• Added a description of numeric value notations.
1.5.0	01/31/2005	• Added a description of the `printvar` command. • Changed "NITRO" to "Nintendo DS".
1.4.1	11/10/2004	• Explained difference between `volume` and `volume2` commands in detail. • Explained `randvar` command in detail. • Fixed typos.
1.4.0	10/12/2004	• Added an explanation of the method for specifying a player with a label in the `@SEQ_TABLE` sequence archive.
1.3.0	09/16/2004	• Made it possible to use a value of 0 for the argument of the `loop_start` command so that it is handled as an infinite loop. • Made it possible to use `loop_start` and `loop_end` from an SMF. • Made it possible to use `loop_start` and `loop_end` for loop markers in an SMF. • Revised the description of Blank Space at Start of SMF. • Made the correction in "Compatibility with SMF created for MusicPlayer2000" because of the addition of the loop feature for individual tracks in an SMF. • Added the description on how to specify banks by using labels with `@SEQ_TABLE` of sequence archive.
1.2.0	09/02/2004	• Added the description about embedding text commands in SMF.
1.1.0	08-10-2004	• Revisions reflecting changes in the `smfconv` output format.
1.0.0	07/20/2004	• Style adjustments
0.5.0	06/01/2004	• Added expansion sequence command. • Added list of all sequence commands. • Added table for MIDI control codes. • Added a description of how to specify an index number for the sequence label name of the sequence archive. • In conjunction with the change of being able to play back multiple sequences with one Player, revised the description of Player Numbers in the sequence table of the sequence archive. • Revised the description of the Player priority. • Made MIDI control change (29) to increase the value by 24 time and convert to `sweep_pitch` command. • Changed the maximum value of the `prg` and `mod_delay` commands from 65535 to 32767. • Added `volume2` command. • Clearly stated the range of the `bendrange` command value.
0.4.0	04/12/2004	• Placed the explanation about Sequence Tables in a separate section. • Made distinction for sequence label and data label names. • Added a warning about multiple definition of sequence labels

Version	Revision Date	Description
0.3.0	04/01/2004	• Assignment of `porta` of MIDI Control Change #84. • Added Envelope command. • Added section about compatibility of SMF and MusicPlayer2000. • Added `loop_start` / `loop_end` commands. • Added description that only single sounds can be played in tie mode. • Added a warning about infinite loops with the `jump` command. • Deleted the incorrect statement that `mod_range` worked only for pitch change. • Corrected errors about Voice Priority comparisons.
0.2.0	03/18/2004	• Changed the overall composition. • Made corrections in step with changes to text format specifications. • Added section relating to special specification for when Note command `length` is 0. • Added section relating to expanded specifications for portamento and sweep.
0.1.0	03/01/2004	Initial version.

1   Overview

Sequences are a data format that correspond to sheet music data—such as, a standard MIDI file. Commands that process sound generation and volume change over time are described. Sequence data is a collection of commands, and sounds are generated using bank data that corresponds to the sound source data.

Sequences have two formats: sequence files and sequence archive files.

This section provides simple explanations of both types of files.

1.1   Sequence Files

Sequence files contain one set of sequence data per file. Sequence files can be handled the same way as standard MIDI files.

By using the SMF converter `smfconv,` a standard MIDI file can be converted into sequence files.

1.2   Sequence Archives

A Sequence Archive is a single file that contains a collection of sequence data. `smfconv` cannot be used to convert data to this format; therefore, use a text editor to create a sequence archive. Sequence archives, can reduce the of data size. Furthermore, because many sequences are handled in a single file, simple sequence data can be created in a short time.

2   Basics

2.1   Time Base

The quarter-note resolution is 48. When the tempo is 240, the length of one produced sound is equal to the length of one tick. Variations in sound generation can be minimized by setting the tempo to a divisor of 240. The default tempo is 120.

2.2   Comments

Comments in sequence files and sequence archives start with a semicolon (;) for each line of code written in text format as shown below.

```
;;; comment

Track_0:  ; comment
```

2.3   Labels

A colon (:)at the end of the character string defines a a label definition. Once a label is defined, that label can be called from any other file.

2.3.1   Characters Used for Labels

All global label names muststart with a letter. Subsequent characters can be letters, numbers, or an underscore (_).

```
LoopStart:
test_seq:
Track_01
```

2.3.2   Local Labels

Label names that *begin* with an underscore (_) are treated as local labels.

A local label is valid only in the lable section. Therefore, the same local label name can be used in another section.

```
label_1:
        ;; The "_local" from line 3 can be used here
_local:
        ;; The "_local" from line 3 can be used here

label_2:
        ;; The "_local" from line 7 can be used here
_local:
        ;; The "_local" from line 7 can be used here

label3:
        ;;  The "_local" cannot be used here
```

3   Sequence Files

3.1   Overview

A Sequence file is a file that stores a single sequence. Normally, the sequence file is created by converting a standard MIDI file.

The standard MIDI file is first converted into a text format sequence file. The text format sequence file is implemented using the same sequence commands as a Sequence Archive explained in following sections.

The text sequence file is then converted to the binary data that has the file extension *.sseq by the sequence converter `seqconv` and played on the DS.

3.2   Standard MIDI Files

Standard MIDI files (SMF) are created using commercially available sequencers.

Below are some precautions to take when creating SMF.

3.2.1   SMF Format

Use only SMF format 0 or 1. There is no support for SMF format 2.

3.2.2   Tracks

A maximum of 16 tracks can be used. However, the NITRO-Composer System can use only up to 32 tracks.

Channel numbers 1 to 16 correspond to track numbers 0 to 15. MIDI events like tempo changes that affect the overall sequence get mixed into track 0 for output.

3.2.3   Specifying Loops

To loop all of the tracks in SMF using the same timing, write square brackets as markers on the MIDI sequencer. The file is converted by taking the opening bracket ([ ) as the start of the loop, and the closing bracket ( ]) as the end of the loop. The opening bracket and the closing bracket can be replaced with the "`loop_start`" and "`loop_end`" character strings respectively.

To loop each track, enter a value of 0 for Control Change 89 for the loop starting point; enter an arbitrary value for Control Change 90 (with an arbitrary value) for the loop ending point.

3.2.4   Blank Space at Start of SMF

When the SMF is converted, all empty notes are truncated automatically until the first Note On. To keep the empty notes from being truncated, add a low-volume dummy note to start the sequence.

However, if the start of a loop is placed before the first Note On, only the empty notes before the starting position of the marked loop are truncated.

3.2.5   MIDI Events

To learn about the MIDI events that are converted, see the list of sequence commands in 5 - Sequence Commands.

3.2.6   Compatibility with MusicPlayer2000

An SMF that is created for MusicPlayer2000 (the sound driver for AGB) can be converted without modification. However, some of the specifications have changed, and adjustments are needed to reproduced the file correctly. For details, see 7.3 - -Compatibility with SMF Created for MusicPlayer2000.

3.3 Text Sequences

3.3.1 Overview

When the SMF is converted using smfconv, the text sequence file shown in Code 3-1 is generated.

Code 3-1 Text Sequence File

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; mid/kart64_title.smft
;    generated by smfconv
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SMF_kart64_title_Begin:
      alloctrack 0x0eff
SMF_kart64_title_Start:
      opentrack 1, SMF_kart64_title_Track_1
      opentrack 2, SMF_kart64_title_Track_2
      opentrack 3, SMF_kart64_title_Track_3
      opentrack 4, SMF_kart64_title_Track_4
      opentrack 5, SMF_kart64_title_Track_5
      opentrack 6, SMF_kart64_title_Track_6
      opentrack 7, SMF_kart64_title_Track_7
      opentrack 9, SMF_kart64_title_Track_9
      opentrack 10, SMF_kart64_title_Track_10
      opentrack 11, SMF_kart64_title_Track_11


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Track 0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SMF_kart64_title_Track_0:
   notewait_off
Measure 1 ---------------------------------
   tempo       140
   prg         0
SMF_kart64_title_Track_0_LoopStart:
   wait        1
   volume      127
   pan         64
   bendrange   2
   pitchbend   0
   mod_speed   16
   mod_depth   0
   wait        767
; Measure 2 ---------------------------------
```

This text sequence file is written as a series of sequence commands. Generally, these commands are processed from top to bottom. For explanations of each command, see 5 - Sequence Commands.

3.3.2 Track Opening Process

The process to open tracks follows the first comment..

Code 3-2   The track opening process

```
SMF_kart64_title_Begin:
      alloctrack 0x0eff
SMF_kart64_title_Start:
      opentrack 1, SMF_kart64_title_Track_1
      opentrack 2, SMF_kart64_title_Track_2
      opentrack 3, SMF_kart64_title_Track_3
      opentrack 4, SMF_kart64_title_Track_4
      opentrack 5, SMF_kart64_title_Track_5
      opentrack 6, SMF_kart64_title_Track_6
      opentrack 7, SMF_kart64_title_Track_7
      opentrack 9, SMF_kart64_title_Track_9
      opentrack 10, SMF_kart64_title_Track_10
      opentrack 11, SMF_kart64_title_Track_11
```

Keep in mind that the first set of sequence data is processed on track 0. When the sequence starts, track 0 is already allocated, and the process to open sequences begins at that point. When using multiple tracks, another track needs to be allocated and started from track 0.

`alloctrack` on line 2 allocates tracks. This command can be used only immediately after the start of the sequence. The argument is a bitmask expressing the tracks starting at track 0 from the LSB to allocate the bit enabled track.

`opentrack` from lines 4 and on starts the tracks. The first argument is the track number; the second argument is the label for the beginning of the sequence on that track.

From the position of this label, the process on the specified track begins. The processes for track 0 executes the lines that follow after the last `opentrack` command.

3.3.3   Track Process

The process for each track is described after the label as seen in Code 3-1 for `kart64_title_Track_0:`.

Code 3-3   The track process

```
SMF_kart64_title_Track_0:
  notewait_off
; Measure 1 --------------------------------
    tempo       140
    prg         0
SMF_kart64_title_Track_0_LoopStart:
    wait        1
    volume      127
    pan         64
    bendrange   2
    pitchbend   0
    mod_speed   16
    mod_depth   0
    wait        767
; Measure 2 --------------------------------
```

For more information about the each sequence command, see 5 - Sequence Commands.

3.4 Embedding text commands in SMF

3.4.1 Overview

Creating sequence data on a sequencer is convenient because you can check the sound immediately. There are many restrictions with SMF; furthermore, most commands used with text sequences cannot be used with SMF. You can edit converted `*.smft` files directly with a text editor to use these commands , but the file will be overwritten the next time you edit the file with the sequencer.

For this reason, a feature is provided to embed text commands in SMF and output text commands and MIDI events in a *.smft file. With this feature, any text command can be used and the command will not be overwritten when the file is edited with the sequencer.

3.4.2 How to Embed

Embed the following character string in the SMF data as either a marker or a text event.

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

This character string outputs the command "`pan_r 30, 90`" at the specified location on Track 03 (MIDI channel).

3.4.3 Output All Tracks

as the statement shown below, is specified for all tracks.

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

However, this statement does not output information for unused tracks.

3.4.4 Expansion of Label Name

As shown below, defining the label in the same place on all tracks using "`text_all`" causes an error.

```
text_all:BLOCK_A:
```

This statement defines label `BLOCK_A` at multiple locations, and causes a multiple definition error. In this case, use the following statement.

```
text_all:$BLOCK_A:
```

By adding the `$` character, the label name is given a unique prefix and passed into each track as shown below.

```
SMF_filename_Track_0_BLOCK_A:
```

`filename` indicates the SMF file name `0` indicates the track number. and the edition of this information prevents the multiple definition error.

3.4.5   Cautions regarding the real-time MIDI playback

Real-time MIDI playback does not process the embedded text sequence. For this reason, the result is different from the playback of the converted file.

4   Sequence Archives

4.1   Overview

A Sequence Archive is a single file that comprises multiple sets of sequence data. The example shown in Code 4-1 helps explain sequence archives.

Code 4-1   Text Sequence Archive

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;      SeqArc for Sample SE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

#include "../sound_data.sbdl"

@SEQ_TABLE

SE_YOSHI:              yoshi,          BANK_SE,  127, 96, 64, PLAYER_VOICE
SE_WIHAHO:             wihaho,         BANK_SE,  127, 96, 64, PLAYER_VOICE
SE_COIN:               note_only,      BANK_SE,   65, 96, 64, PLAYER_SE
SE_AMBULANCE:          jump_seq,       BANK_SE,   55, 96, 64, PLAYER_SE
SE_PATTERN:            loop_seq,       BANK_SE,   55, 96, 64, PLAYER_SE
SE_PATTERN:            call_seq,       BANK_SE,   55, 96, 64, PLAYER_SE
SE_PORTAMENT:          porta_seq,      BANK_SE,   65, 96, 64, PLAYER_SE
SE_PORTAMENT2:         porta_time_seq, BANK_SE,   65, 96, 64, PLAYER_SE
SE_SWEEP:              sweep_seq,      BANK_SE,   65, 96, 64, PLAYER_SE
SE_VIBRATE:            mod_seq,        BANK_SE,   65, 96, 64, PLAYER_SE
SE_VIBRATE2:           tie_seq,        BANK_SE,   65, 96, 64, PLAYER_SE
SE_SUPER_MARIO:        waitoff_seq,    BANK_SE,   65, 96, 64, PLAYER_SE
SE_SUPER_MARIO2:       opentrack_seq,  BANK_SE,   65, 96, 64, PLAYER_SE

@SEQ_DATA

yoshi:
    prg 0
    cn4 127, 0
    fin

wihaho:
    prg 1
    cn4 127, 0
    fin
```

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


; loop by jump (repeated endlessly)
; Ambulance
jump_seq:
      prg 3
_loop_start:
      bn4 127, 48
      gn4 127, 48
      jump _loop_start
```

4.2 Using the Sound Archive Label File

This section explains the first statement in the file.

`#include "../sound.data.sbdl"`

This statement shows that the sound archive label file named `sound_data.sbdl` is included in the directory a level above the current directory. The sound archive label file is generated automatically during sound data conversion. By including this file, the labels defined in the sound archive definition file can be used in this sequence archive.

By including this statement at the beginning of the file, the bank label can be used for specifying banks in subsequent lines. Including this file with `#include` is unnecessary if you specify banks with numbers instead of labels.

4.3 Structure

The structure of the code is divided into two parts:

- Sequence table
- Sequence data

4.3.1   Sequence Table

Code 4-2 show a sequence table.

Code 4-2   Sequence Table

```
@SEQ_TABLE

SE_YOSHI:               yoshi,          BANK_SE, 127, 96, 64, PLAYER_VOICE
SE_WIHAHO:              wihaho,         BANK_SE, 127, 96, 64, PLAYER_VOICE
SE_COIN:                note_only,      BANK_SE, 65, 96, 64, PLAYER_SE
SE_AMBULANCE:           jump_seq,       BANK_SE, 55, 96, 64, PLAYER_SE
SE_REPEAT:              loop_seq,       BANK_SE, 55, 96, 64, PLAYER_SE
SE_PATTERN:             call_seq,       BANK_SE, 55, 96, 64, PLAYER_SE
SE_PORTAMENT:           porta_seq,      BANK_SE, 65, 96, 64, PLAYER_SE
SE_PORTAMENT2:          porta_time_seq, BANK_SE, 65, 96, 64, PLAYER_SE
SE_SWEEP:               sweep_seq,      BANK_SE, 65, 96, 64, PLAYER_SE
SE_VIBRATE:             mod_seq,        BANK_SE, 65, 96, 64, PLAYER_SE
SE_VIBRATE2:            tie_seq,        BANK_SE, 65, 96, 64, PLAYER_SE
SE_SUPER_MARIO:         waitoff_seq,    BANK_SE, 65, 96, 64, PLAYER_SE
SE_SUPER_MARIO2:        opentrack_seq,  BANK_SE, 65, 96, 64, PLAYER_SE
```

Calling `@SEQ_TABLE` on line 1 indicates the start of the sequence table. The sequence table data begins on line 3. Each line defines a single sequence. The details about sequence data are explained in the next section.

4.3.2   Sequence Data

Code 4-3 shows sequence data:

Code 4-3   Sequence data

```
; Note commands only
; Coin sound
note_only:
      prg 0
      as5 127, 6
      ds6 127, 48
      fin

; loop by jump (repeated endlessly)
; Ambulance
loop_seq:
      prg 1
_loop_start:
      bn4 127, 48
      gn4 127, 48
      jump _loop_start
```

Each sequence begins with a label name definition, and the sequence associated with that label. The label name definition is followed by sequence commands used to create a single set of sequence data. For further details about each sequence command, see 5 - Sequence Commands.

4.4 Sequence Tables

This section discussed the contents of a sequence table.

Code 4-4   Sequence table

```
@SEQ_TABLE

SE_YOSHI:              yoshi,          BANK_SE, 127, 96, 64, PLAYER_VOICE
SE_WIHAHO:             wihaho,         BANK_SE, 127, 96, 64, PLAYER_VOICE
SE_COIN:               note_only,      BANK_SE, 65, 96, 64, PLAYER_SE
SE_AMBULANCE:          jump_seq,       BANK_SE, 55, 96, 64, PLAYER_SE
SE_REPEAT:             loop_seq,       BANK_SE, 55, 96, 64, PLAYER_SE
SE_PATTERN:            call_seq,       BANK_SE, 55, 96, 64, PLAYER_SE
SE_PORTAMENT:          porta_seq,      BANK_SE, 65, 96, 64, PLAYER_SE
SE_PORTAMENT2:         porta_time_seq, BANK_SE, 65, 96, 64, PLAYER_SE
SE_SWEEP:              sweep_seq,      BANK_SE, 65, 96, 64, PLAYER_SE
SE_VIBRATE:            mod_seq,        BANK_SE, 65, 96, 64, PLAYER_SE
SE_VIBRATE2:           tie_seq,        BANK_SE, 65, 96, 64, PLAYER_SE
SE_SUPER_MARIO:        waitoff_seq,    BANK_SE, 65, 96, 64, PLAYER_SE
SE_SUPER_MARIO2:       opentrack_seq,  BANK_SE, 65, 96, 64, PLAYER_SE
```

The format of each statement is shown below:

```
seqName: dataLabel, bankNo, volume, channelPrio, playerPrio, playerNo
```

The description of the statement elements is shown in Table 4-1.

Table 4-1 Sequence Table Elements

Element	Description
seqName	Sequence label name
dataLabel	Data label name
bankNo	Bank label or bank number
Volume	Volume
channelPrio	Voice priority
playerPrio	Player priority
playerNo	Player label or player number

The following sections discuss each element.

4.4.1   Sequence Label Name

The sequence label name identifies a specific sequence. The name must begin with an uppercase
Roman letter but subsequent characters can be uppercase Roman letters, underscores, and numbers.
The same label name cannot be defined more than once because label names are global. Note that
the same label name should not be defined even in other sequence archives.

The label name specified here is used in the program to play back the sequence. Instead of the label
name, the index number can be used to specify a sequence.

```
SE_COIN = 2:     note_only,        BANK_SE,  65, 96, 64, PLAYER_SE
SE_AMBULANCE:    loop_seq,         BANK_SE,  55, 96, 64, PLAYER_SE
10:              call_seq,         BANK_SE,  55, 96, 64, PLAYER_SE
```

In the above example, statement 1 specifies both the label and index number. Here, the label is SE_COIN and the index number is 2.

If only a label name is specified, the index number for this statement will be determined by incrementing the index number of the previous statement by one. Therefore, in the second statement in the above example, the index number is 3.

The third statement is assigned an index number only and the label name is undefined. Because this sequence archive is assigned only an index number, the sequence archive must be referred to using the index number.

4.4.2   Data Label Name

The data label name specifies the label name of the sequence data described after @SEQ_DATA.

4.4.3   Bank

Using the label name or number, specifies the bank to use for sequence playback. However, if a sound archive label file (*.sbdl) is not included , a bank cannot be specified with a bank label.

4.4.4   Volume

Volume is specified to adjust the overall sequence volume, and is a values from 0 to 127.

4.4.5   Voice Priority

Specifies priority for sound generation of all sequences.

The voice priority determines which sound to generate from the 16 channels. The value ranges from 0 to 127, and the higher the value, the higher the priority.

For details on Voice priority, see the Sound System Manual.

4.4.6   Player Priority

This specifies the player priority.

The player priority determines which sequence will play from the maximum number of sequences that can play. The value ranges from 0 to 127, and the higher the value, the higher the priority.

For details on Player Priority, see the Sound System Manual.

4.4.7   Player Number

Using the play label or player number, player number specifies which of the 32 players to use for playback. However, you can specify the player label only if the sound archive label file (*.sbdl) is loaded.

The number of sequences that can play back on a single Player is limited; therefore, the player must

be specified if this limitation is taken into consideration..

4.5   Numeric Value Notation

The numeric value notations in the following sections can be used in place of decimal values to specify numeric parameters in the text sequence file.

4.5.1   Binary and Hexadecimal Notation

Numeric values are frequently expressed as decimals, but can also be expressed in binary and hexadecimal notations.

When describing binary or hexadecimal numbers, ,attach the prefix `0b` or `0x` respectively to begin the notation. For example, the decimal number 12 is expressed as:

```
0b1100
0xc
```

4.5.2   Bit Notation

Bit notation is helpful when writing a value that specifies whether a bit is 1 or 0, such as with a bit flag.

Bit notation expresses which bits are set to 1. For example, when setting bits 1, 3, and 6 to 8 to the value of 1, it should be written as:

```
{ 1,3,6-8 }
```

This corresponds to 0b111001010. Note that the LSB is 0.

4.5.3   Mathematical Expressions

Numeric values may also be written as mathematical expressions. Binary, hexadecimal, and bit notations may be used in the terms of the mathematical expression.

For example, the following notations are possible:

```
2 * 4 + 0x10
( 1 << 4 ) + 3
{ 0, 2 } | { 4-6 }
```

Table 4-2 shows the operators that can be used in an expression, as well as the priority of the operators.

Table 4-2 Operators

Priority	Operator	Meaning
1	*	Multiplication
	/	Division
2	+	Addition

	–	Subtraction
3	>>	Right Shift
	<<	Left Shift
4	<	Left side is Less Than the right side
	<=	Left side is Less Than or Equal To the right side
	>	Left side is Greater Than the right side
	>=	Left side is Greater Than or Equal To the right side
5	==	Left side is equal to the right side
6	&	Bitwise AND
7	|	Bitwise OR

5   Sequence Commands

Sequence Commands are the group of commands used in the sequence data.

Table 5-1 List of Sequence Commands

Command Name	Explanation	Default	MIDI	Ref.(pg)
`cn4 velocity, length`	Note command		`$9x, $8x`	25
`wait length`	Rest command			26
`fin`	End sequence			26
`prg x`	Change program	0	`$Cx`	26
`tempo x`	Change tempo	120		26
`volume x`	Change track volume	127	`$Bx, 7`	27
`volume 2x`	Change track volume	127	`$Bx, 11`	27
`main_volume x`	Change Player's volume	127	`$Bx, 12`	27
`pitchbend x`	Change pitch bend	0	`$Ex`	27
`bendrange x`	Change pitch bend range	2	`$Bx, 20`	27
`transpose x`	Transpose	0	`$Bx, 13`[i]	27
`pan x`	Change track pan	64	`$Bx, 10`	28
`prio x`	Change voice priority of track	64	`$Bx, 14`	**Error! Bookmark not defined.**
`tieon / tieoff`	Tie mode start/end	off		28
`notewait_on / notewait_off`	Note wait on/off	on		28
`porta x`	Set portamento's start key and start	cn4(60)	`$Bx, 84`[ii]	29
`porta_time x`	Set portamento time	0	`$Bx, 5`	29
`porta_on / porta_off`	Portamento start/end	off	`$Bx, 65`[iii]	29

---

[i] 0 to 127 for transpose (#13) becomes -64 to +63.
[ii] For portamento control (#84), specify the note number that takes 60 as cn4 (middle C).
[iii] 0 to 63 for portamento (#65) becomes `porta_off`, while 64 to 127 become `porta_on`.

Command Name	Explanation	Default	MIDI	Ref.(pg)
sweep_pitch x	Set the amount-of-change in sweep pitch	0	$Bx, 28^{iv} $Bx, 29^{v}	30
alloctrack mask	Secure tracks			30
opentrack no, label	Start a track			30
jump label	Sequence jump			30
call label	Sequence call			31
ret	Sequence return			31
loop_start count	Loop starting point		$Bx, 89	31
loop_end	Loop ending point		$Bx, 90	31
attack x	Set attack value		$Bx, 85	31
decay x	Set decay value		$Bx, 86	31
sustain x	Set sustain value		$Bx, 87	31
release x	Set release value		$Bx, 88	31
envelope a,d,s,r	Set envelope values			31
mod_depth x	Set modulation depth	0	$Bx, 1	32
mod_range x	Set modulation range	1	$Bx, 23	32
mod_speed x	Set modulation speed	16	$Bx, 21	32
mod_delay x	Set modulation delay	0	$Bx, 26 $Bx, 27^{vi}	32
mod_type x	Set modulation type	0	$Bx, 22	32

5.1 Note command

This generates sounds using the current program number.

```
cn4 velocity, length
```

Keys are written in the format cn4, cs4, dn4, ds4, en4, fn4, fs4, gn4, gs4, an4, as4, bn4, and cn5. The middle C is cn4. The value ranges from cnm1 to gn9 can be specified. (m1 indicates "minus 1.")

velocity specifies a value that ranges from 0 to 127, and the value is based on a squared value scale.interpreted. In other words, by taking127 as 100%:

[iv] 0 to 127 for sweep pitch (#28) becomes -64 to +63.
[v] With 0 to 127 for sweep pitch (#29), the values are multiplied by 24 and become –1536 to +1512.

[vi] Modulation delay (27) multiples the value by 10 for output.

```
( ( velocity / 127 ) ^ 2 ) * 100 [%]
```

`length` indicates the length of a note in which 48 corresponds to the length of a quarter note. The actual length depends on the tempo. The value ranges from 0 to 268435455. The value 0 represents infinity. In other words, the waveform data plays to the end. If the waveform data loops, the sound plays back indefinitely (the playback can be stopped the program).

In the note-wait ON state (default), the sequence stops for the same length of time as the length of a note. When `length` is 0, the sequence waits for playback to end in the channel. In other words, the next sequence process starts after the waveform data has played back to the end. If the waveform data loops, the sequence stops indefinitely (can be stopped in the program). Even if the note has low voice priority, be aware that the process shifts to the next sequence, ,and the playback is forcibly stopped.

Code 5-1   Example of Note command

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

5.2 Rest Command

This command stops the sequence for a specified length of time.

```
wait length
```

`length` matches the length of a note, where 48 corresponds to the length of a quarter note. The actual length depends on the tempo. The value ranges from 0 to 268435455.

5.3 End Sequence

This command ends the sequence process of the track.

```
fin
```

When the sequence processes for all tracks end, the Player processes also stop. If you forget to write the `fin` command, the following sequence data will execut without a break. This oversight can cause unexpected sounds to play and the system to hang.

5.4 Change Program

This command executes a program change.

```
prg x
```

A program number ranges from 0 to 32767. However, only values from 0 to 127 can be specified with MIDI. The default value is 0.

5.5 Change Tempo

This command changes the tempo.

```
tempo x
```

A tempo value ranges between 0 and 1023. The tempo scale that can be set in the program determines the ultimate tempo value. The default value is 120.

5.6 Track Volume

This command changes the track volume.

```
volume x
volume2 x
```

A `volume` value i ranges from 0 to 127. The default value is 127.

Like `velocity`, the `volume` value is based on a squared value scale. In other words, if 127 is taken as 100% :

```
( ( volume / 127 ) ^ 2 ) * 100 [%]
```

There is no functional difference between `volume2` and `volume`. When both are specified at the same time, both values are reflected in the results. For example, if `volume` is set to 80% and `volume2` is set to 50%, the volume will be 40% (`0.8 x 0.5 = 0.4`).

5.7 Main Volume

This command changes the volume of the player.

```
main_volume x
```

The `volume` value ranges from 0 to 127. The default value is 127.

Like `velocity`, the `volume` value is based on a squared value scale. In other words, if 127 is taken as 100%:

```
( ( volume / 127 ) ** 2 ) * 100 [%]
```

5.8 Pitch bend

Changes the pitch bend and the pitch bend range.

```
pitchbend x
bendrange x
```

The pitch bend value ranges from `-128` to `127`. The default value is `0`.

The units of `bendrange` aresemitones. The default value is `2` and the value ranges from `0` to `127`.

When the maximum `pitchbend` value is specified, the pitch of the note changes by the value specified by `bendrange`. For example, if the value assigned to `bendrange` is `2`, a `pitchbend` value of `127` changes by +2 semitones, a value of `64` changes by +1 semitone, and the value of `-64` changes by -1 semitone.

5.9 Transpose

Transposes the sequence..

```
transpose x
```

The `transpose` value specified in units of semitones ranges from -64 to +63. The default value is 0.

5.10 Track Pan

Changes the pan for the track.

```
pan x
```

The pan value ranges from 0 and 127, with 0 being left, 127 being right, and 64 being center. The default value is 64.

5.11 Voice Priority

This command sets the voice priority for the track.

```
prio x
```

The priority value ranges from 0 to 127. The default value is 64. In actuality, the final voice priority value comprises the player and voice priority. A larger value indicates higher priority. When both values match, player priority has priority to generate sound.

5.12 Tie Mode

Changes the tie mode.

```
tieon
tieoff
```

The default is tie mode off.

When Note On is generated while tie mode is on, regardless of the note length specified by the Note command, the sound will play continuously. When the next Note On is generated, sound will continue to generate but only the pitch and velocity will change. To discontinue generating sound, set the tie mode to off or stop the sequence.

When tie mode is set to on, sounds being generated are forcibly released. Even if note-wait mode is set to `off` , note that only single notes can be played.

5.13 Note Wait Mode

Changes the note-wait mode

```
notewait_on
notewait_off
```

By default, note-wait is ON.

When Note On is generated and note wait is ON, the sequence stops for the same length of time as

the note length specified by the Note command. When note wait is OFF, the next command is processed without a break.

In the ON state, the sequence stops while notes are playing and not needing to insert a Rest command is convenient.. However, nothing can be processed while the notes are playing. In the `off` state, there is no wait after a Note command, so two or more notes can play atonce and move the pan while notes are playing.

5.14 Portamento

Sets the portamento.

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

Enters the portamento mode with the porta command. When a `Note` command is executed in portamento mode, the pitch specified with `key` changes to a pitch specified by the `Note` command during playback. With following calls to the `Note` command, the previous `Note` command pitch changes to the current `Note` command pitch.

The `porta_time` command specifies the speed of the pitch change. (Following the MIDI convention, this command is called `porta_time` instead of `porta_speed`.) The value ranges from 0 to 255, and the default value is 0. The larger the value, and the change takes longer. When set to 0, the speed of the pitch change is the same length as the note length. In other words, the pitch changes during the time set to generate sound using the note command.

`porta_on` like `porta` enter portamento mode, but does not specify a starting `key`. With the next note command, the pitch is changed from the pitch specified in the note command executed before `porta_on`. When the next note command is executed, the pitch changes from the previous note command to the pitch of the current note command.

The `porta_off` command cancels the portamento mode.

If portamento is used while in the `tieon` state, commands such as repeatedly changing the pitch with pitchbend are possible.

5.15 Sweep

Configures the sweep settings.

```
sweep_pitch pitch
```

When the `sweep_pitch` command is used, play the sound with a sweeping effect. When `Note` is executed while the sweep is set, the sound begins offset by the value `pitch` from the pitch specified by `Note`; the pitch is then gradually changed to the correct pitch. The `pitch` value rangesfrom -32768 to 32767, and the default value is 0. A value of 64 shifts the pitch by exactly one semitone.

The speed it takes to return to the correct pitch is the same speed set in the `porta_time` command.

5.16 Allocate Tracks

Allocates the tracks.

```
alloctrack mask
```

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

`mask` is the bit mask of the tracks that are going to be secured. Starting from the LSB, the bit masks represent track0, track1, track2, etc. The tracks with enabled bits are allocated. (Track0 is already allocated, so the LSB is ignored.) Using the macro, this command can be written as:

```
alloctrack TRACK_1 | TRACK_2 | TRACK_3
```

Using the above statement, allocates track1, track2, and track3.

5.17 Start a Track

Starts a different track.

```
opentrack no, label
```

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

`no` is the track number, and the value ranges from 0 to 15, but the current track cannot be specified. The track being executed will execute once all of the notes are released.

`label` indicates the start of the sequence.

If a track with a larger number than the current track is opened, the first process is executed in the same frame. However, if the track with a smaller number is opened, the process is executed one frame later.

5.18 Sequence Jump

Jumps from one location in the sequence to another.

```
jump label
```

`label` indicates the jump destination.

By using jump, a loop sequence can be created. However, if commands that stop sequences are not included in the loop--such as the wait or note command with the note wait on-- the sequence loops indefinitely and causes an unexpected error.

5.19 Sequence Call

Jumps from one location in the sequence to another, and then returns to the original location.

```
call label
ret
```

`label` indicates the jump destination.

By using the `ret` command at the jump destination, the sequence will return to the original location of the call. `call` can be executed again at the jump destination, but `call` can be called only up to three levels.

Note that these three levels include the nested levels explained in 5.20 - Loop.

5.20 Loop

Specifies the number of times to execute a certain section of a sequence repeatedly.

```
loop_start count
loop_end
```

`loop_start` sets the starting point of the loop. `count` sets the number of loops, and the values ranges from 0 to 255. When the value is 0, the loop is infinite. `loop_end` sets the ending point of the loop. The process returns to the starting point of the loop until it reaches the number of loops specified by `count`.

Embedded loops can be set within other loops, but only up to three levels. Note that three levels include the nested levels explained in 5.19 - Sequence Call.
Note that if this command is used with MIDI during the real-time MIDI playback, loops are ignored .

5.21 Envelope

Configures the envelope settings.

```
attack x
decay x
sustain x
release x
envelope a, d, s, r
```

The bank's envelope values are used by default, but the envelope values can be overwritten by using these commands. If the value for each individual command is specified, the unspecified values are set with the bank's envelope values. Using the `envelope` command, all values can be set at once.

5.22 Modulation

Configures the modulation settings.

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

Set the depth of the modulation with `mod_depth`. The value Ranges from 0 to 127, and the default value is 0.

The maximum values for the degrees of change are shown in Table 5-2. (If mod_range described later is set to 1.)

Table 5-2 Degrees of Modulation Change

Type	Degrees of Change
Change in pitch	Changes between ±1 asemitone
Change in volume	Changes between ±6.0 dB
Change in location	Changes between Left MAX to Right MAX

`mod_range` sets the maximum variation width of change. The value ranges from 0 to 127, and the default value is 1. For example, if the pitch changes with the default setting, the change occurs in the range of ±1 semitone at the maximum mod_depth and a mod_range of 12; the modulation occurs in the range of ± 12 semitones, or one octave.

`mod_speed` sets modulation speed. The value ranges from 0 to 127, and the default value is 16. When the value is set to 1, the modulation speed linearly changes at approximately 0.4Hz, and the value that can be set ranges from approximately 0.0Hz to 50Hz.

`mod_delay` sets the delay time of the modulation. The value ranges from 0 to 32767, and the default value is 0. The delay time is measured in units of sound frames (approx. 5 ms), and is independent from the tempo. Use Control Change 26 or 27 with MIDI. Control Change 26 outputs a value as-is; the Control Change 27 outputs a value multiplied by 10. Consequently, the value that can be set ranges from 0 to 1270.

`mod_type` sets modulation type. The possible settings are shown in Table 5-3. The default value is 0 = vibrato (pitch change).

Table 5-3 List of Modulation Types

Numerical Value	Label	Explanation
0	MOD_TYPE_PITCH	Vibrato (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 volume is already at the maximum when trying to change it, the volume will not become louder; instead, it will only become softer.

6   Extended Sequence Commands

6.1   Overview

The extended sequence commands allow you to configure settings with a higher degree of freedom than the sequence commands described in previous chapters.

For normal sequence playback, normal sequence commands are sufficient. But the use of extended sequence commands allow for more elaborate sequences and interactive sequence playback among other enhancements.

6.2   Random Command

With some sequence commands, random numbers can be set for the values.

```
pitchbend_r -12 , 12
```

In the example above, `pitchbend` is given a random value range of `-12` to `+12`. This command allows for subtle changes to the pitch each time a sound plays.

6.2.1   Format

The basic format of the random command is shown below.

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

The random numbers are generated between the minimum value of `min` and the maximum value of `max`. These values are executed as the arguments of the `command`.

6.2.2   Random Commands

Table 6-1 List of Random Commands

wait_r	prg_r	volume_r	volume2_r
main_volume_r	pitchbend_r	pan_r	transpose_r
porta_time_r	sweep_pitch_r	mod_depth_r	mod_speed_r
attack_r	decay_r	sustain_r	release_r
mod_delay_r	loop_start_r	setvar_r	addvar_r
subvar_r	mulvar_r	divvar_r	shiftvar_r
randvar_r	cmp_eq_r	cmp_ge_r	cmp_gt_r
cmp_le_r	cmp_lt_r	cmp_ne_r	

The `setvar_r` command and subsequent commands in Table 6-1 are explained later.

A table that shows the correspondence with the normal sequence commands can be found in 7.1 Complete List of Sequence Commands.

6.2.3   Random Note Length Command

Adding an `_r` to a note command makes it a random command. This command sets the note length to a random range of values.

```
cn4_r velocity min, max
```

To change the pitch randomly, use `transpose_r`. To change the velocity randomly, use `volume_r` or `volume2_r`.

6.3   Variable Commands

Variables can be handled in a sequence.

6.3.1   What is variable?

Variables are locations in memory that allows the storage of numeric values. In one variable values ranging from –32768 to 32767 can be stored.

16 variables can be used in one sequence and 16 variables can be shared by the entire system. Variables handled only in one sequence is called a local variable, and variables hared by the entire system is called a global variable.

Figure 6-1 Local and Global Variables
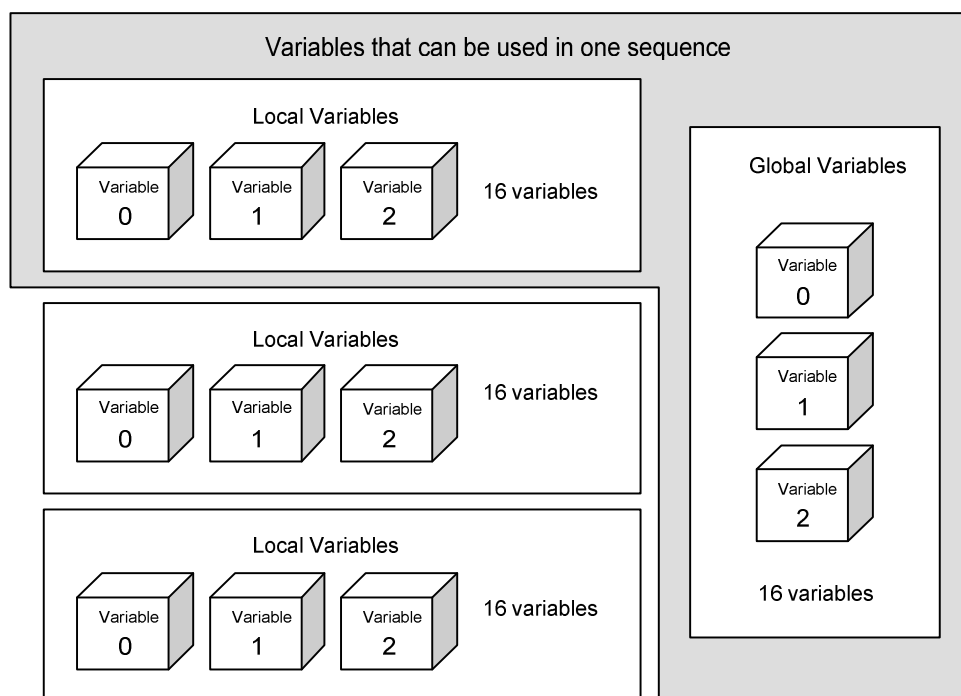
Each variable is specified by numbers in the sequence. Specifying a value from 0 to 15 declares one of the sixteen local variables. Specifying a value from 16 to 31 declares one of the sixteen global variables. In other words, 16 to 31 express the global variables 0 to 15.

The initial value of the variable is –1. The global variables are initialized during the startup of the system, and local variables are initialized when a sequence starts.

6.3.2   Variable Calculation Command

Values can be set and four arithmetic operations can be performed for each variable.

Table 6-2 List of Variable Calculation Commands

Command Name	Description
`setvar varNo, x`	Sets x in the variable
`addvar varNo, x`	Adds x to the variable
`subvar varNo, x`	Subtracts x from the variable
`mulvar varNo, x`	Multiplies x times the variable
`divvar varNo, x`	Divides the variable by x
`shiftvar varNo, x`	Left shifts the variable by x bits (right shift for a negative value)
`randvar varNo, x`	Sets the random value between 0 and x in the variable (or in the range of x to 0 if the value is negative)

6.3.3   Variable Commands

Sequence commands can be executed using set variables with the variable calculation commands listed in Table 6-2.

```
pitchbend_v 0
```

As shown in the above example, the `pitchbend` command is executed using the local variable 0.

6.3.4   Format

The basic format of the variable commands is shown below.

```
(command)_v varNo
```

Use the variable specified with `varNo` to execute the `command`.

The range values are set in the arguments of each sequence command. Note that if values exceed the range set in the variables, proper operation of variable commands cannot be guaranteed.

6.3.5   List of Variable Commands

Sequence commands that can be used as random commands can also be used as variable commands.

Table 6-3 List of Variable Commands

wait_v	prg_v	volume_v	volume2_v
main_volume_v	pitchbend_v	pan_v	transpose_v
porta_time_v	sweep_pitch_v	mod_depth_v	mod_speed_v
attack_v	decay_v	sustain_v	release_v
mod_delay_v	loop_start_v	setvar_v	addvar_v
subvar_v	mulvar_v	divvar_v	shiftvar_v
randvar_v	cmp_eq_v	cmp_ge_v	cmp_gt_v
cmp_le_v	cmp_lt_v	cmp_ne_v	

Commands that begin with cmp_ in Table 6-3 are comparison commands and will be described in 6.4.1 - Comparison Command.

A table that shows the correspondence with the normal sequence commands can be found in 7.1 - Complete List of Sequence Commands.

6.3.6   Variable Note Length Command

Adding a _v to a note command makes it into a variable command. In doing so, the note length is specified with a variable.

```
cn4_v velocity varNo
```

To specify pitch with a variable, use transpose_v. To specify the velocity with a variable, use volume_v or volume2_v.

6.3.7   Calculation Between Variables Command

The variable calculation commands described above add or subtract a certain value to or from a particular variable. By using variable commands such as setvar_v, variables can be assigned to other variables or add the value of a variable to another variable.

Code 6-1 Example of Calculation Between Variables

```
setvar_v 0, 1  ; Assign a value from variable 1 to variable 0
addvar_v 2, 3  ; Add value of variable 3 to variable 2
```

6.4 Conditional Commands

Executing a sequence command can be determined by conditional statements using variables.

6.4.1 Comparison Command

Before using conditional commands, execute comparison commands. Conditional commands excute based on the results of comparison command; comparison commands compare two values to determine if the conditional command is executed..

Table 6-4 lists the available comparison commands:

Table 6-4 List of Comparison Commands

Command Name	Formula	Description
cmp_eq varNo, x	(variable) == x	If the value of the variable and x are equal, true (abbr. for equal)
cmp_ge varNo, x	(variable) >= x	If the value of the variable is greater than or equal to x, true (abbr. for greater than or equal)
cmp_gt varNo, x	(variable) > x	If the value of the variable is greater than x, true (abbr. for greater than)
cmp_le varNo, x	(variable) <= x	If the value of the variable is less than or equal to x, true (abbr. for less than or equal)
cmp_lt varNo, x	(variable) < x	If the value of the variable is less than x, true (abbr. for less than)
cmp_ne varNo, x	(variable) != x	If the value of the variable and x are not equal, true (abbr. for not equal)

If the result of the comparison is true, the following conditional commands are executed. If the result of the comparison is false, the conditional commands are not executed.

6.4.2 Conditional Command

There are corresponding conditional commands for all sequence commands. The conditional command has an `_if` attached to the end of the original command name.

```
pitchbend_if +48
```
In the following example, a random number determines if a sound is played.

Code 6-2 Example of Conditions Command
```
randvar 0, 100   ; Set random number value from 0 to 100
cmp_le 0, 80     ; True if random number value is 80 or lower
cn4 96, 12
dn4_if 96, 12    ; Plays sound only when true
en4 96, 12
fin
```

6.5 Communication with the Program

Communication with the program can be conducted using a variable. The variable can read and write from the program, which utilizes this feature to exchange information.

6.5.1   Communication with the MIDI Sequence

Using a variable, information from the MIDI sequence notifies the program.

The four control changes 16 to 19 are converted to the `setvar` command. Control change 16 is set in the local variable 0. The following control changes, 17, 18, and 19 are set in local variables 1, 2, and 3, respectively.

For example, if a variable is set at each break in the blocks that make up a song, finding which block is currently playing back in the program can be found.

6.5.2   Combination with Conditional Commands

Combining the conditional commands and variables from the program allows you to change a sequence with the program timing.

Code 6-3 Example of Combination with Conditional Command

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

In Code 6-3, the notes C/D/C/D will continue to play if the program does nothing, but if 1 is assigned to local variable 0 in the program, the notes will change and play C/E/C/E.

6.6   Debugging Variables

Operations may not always proceed as expected because of mistakes in the process to perform complex variable operations. The sequence command is provided to check and ensure that the intended value is assigned to the used variable.

Table 6-5 A List of Variable Debug Commands

Command Name	Description
printvar *varNo*	Outputs the value of the variable for debugging

If this command is processed, an output for debugging the following statement will result:

```
#0[1]: printvar No.0 = 1
```

What follows `printvar` indicates that the value of variable number 0 is 1.

`#0[1]` indicates that the player identification number is 0 and the track number is 1. With the player identification number, the numeric value itself does not have significance. It simply shows that the output is from a different sequence if the player identification number differs. The track number shows which track from the sequence was output.

This sequence command is valid only when playing back a sequence on the NITRO-Player or SoundPlayer. This sequence command is ignored when playback is performed using any other method.

The print statement is output to the MCS server or the IS-NITRO-DEBUGGER output window.

7   Appendix

7.1   Complete List of Sequence Commands

Table 7-1 Complete List of Sequence Commands

Command name	Description	Specified Value	Range	Variable[vii]	Ref.
cn4 velocity, length	Note command			O	25
wait x	Wait command			O	26
fin	End sequence				26
prg x	Change program	0	0 - 32767	O	26
tempo x	Change tempo	120	1 - 1023		26
volume x	Track volume	127	0 - 127	O	27
volume2 x	Track volume (expression)	127	0 - 127	O	27
main_volume x	Main volume	127	0 - 127	O	27
pitchbend x	Pitch bend	0	-128 - 127	O	27
bendrange x	Pitch bend range	2	0 - 127		27
transpose x	Transpose	0	-64 - 63	O	28
pan x	Track pan	64	0 - 127	O	**Error! Bookmark not defined.**
prio x	Track voice priority	64	0 - 127		28
tieon / tieoff	Tie mode on/off	off			28
notewait_on / notewait_off	Note wait on/off	on			29
porta x	Set portamento start key and portamento on	cn4(60)	0 - 127		29
porta_time x	Portamento time	0	0 - 255	O	29

[vii] Shows whether it can also be used as a variable and random command

Command name	Description	Specified Value	Range	Variable[vii]	Ref.
porta_on / porta_off	Portamento on/off	off			30
sweep_pitch x	Sweep pitch	0	-32768 - 32767	O	30
alloctrack mask	Secure a track				30
opentrack no, label	Start a track				30
jump label	Sequence jump				31
call label	Sequence call				31
ret	Sequence return				31
loop_start x	Loop start		0 – 255	O	31
loop_end	Loop end				31
attack x	Envelope attack		0 – 127	O	31
decay x	Envelope decay		0 – 127	O	31
sustain x	Envelope sustain		0 – 127	O	31
release x	Envelope release		0 – 127	O	31
envelope a,d,s,r	Envelope ADSR				32
mod_depth x	Modulation depth	0	0 - 127	O	32
mod_range x	Modulation range	1	0 - 127		32
mod_speed x	Modulation speed	16	0 - 127	O	32
mod_delay x	Modulation delay	0	0 - 32767	O	32
mod_type x	Modulation type	0	0 - 2		
setvar no, x	Assign variable			O	
addvar no, x	Add variable			O	
subvar no, x	Subtract variable			O	
mulvar no, x	Multiple variable			O	
divvar no, x	Divide variable			O	
shiftvar no, x	Shift variable			O	
randvar no, x	Assign random number			O	
printvar no	Debug Output Variable			O	
cmp_eq no, x	Comparison (==)			O	
cmp_ge no, x	Comparison (>=)			O	
cmp_gt no, x	Comparison (>)			O	

Command name	Description	Specified Value	Range	Variable[vii]	Ref.
`cmp_le no, x`	Comparison (<=)			O	
`cmp_lt no, x`	Comparison (<)			O	
`cmp_ne no, x`	Comparison (!=)			O	

7.2 MIDI Control Code Table

Table 7-2 MIDI Control Code Table

Decimal	Hexadecimal	Command	Ref.	Decimal	Hexadecimal	Command	Ref.
0	00			64	40		
1	01	mod_depth	32	65	41	`porta_on / porta_off`	29
2	02			66	42		
3	03			67	43		
4	04			68	44		
5	05	porta_time	29	69	45		
6	06			70	46		
7	07	volume	27	71	47		
8	08			72	48		
9	09			73	49		
10	0A	pan	28	74	4A		
11	0B	volume2	27	75	4B		
12	0C	main_volume	27	76	4C		
13	0D	transpose	27	77	4D		
14	0E	prio	**Error ! Book mark not defin ed.**	78	4E		
15	0F			79	4F		
16	10	setvar 0	36	80	50		
17	11	setvar 1	36	81	51		
18	12	setvar 2	36	82	52		
19	13	setvar 3	36	83	53		
20	14	bendrange	27	84	54	`porta`	29
21	15	mod_speed	32	85	55	`attack`	31

Decimal	Hexadecimal	Command	Ref.	Decimal	Hexadecimal	Command	Ref.
22	16	mod_type	32	86	56	decay	31
23	17	mod_range	32	87	57	sustain	31
24	18			88	58	release	31
25	19			89	59	loop_start	31
26	1A	mod_delay	32	90	5A	loop end	31
27	1B	mod_delay (x 10)	32	91	5B		
28	1C	sweep_pitch	30	92	5C		
29	1D	sweep_pitch (x 24)	30	93	5D		
30	1E			94	5E		
31	1F			95	5F		
32	20			96	60		
33	21			97	61		
34	22			98	62		
35	23			99	63		
36	24			100	64		
37	25			101	65		
38	26			102	66		
39	27			103	67		
40	28			104	68		
41	29			105	69		
42	2A			106	6A		
43	2B			107	6B		
44	2C			108	6C		
45	2D			109	6D		
46	2E			110	6E		
47	2F			111	6F		
48	30			112	70		
49	31			113	71		
50	32			114	72		
51	33			115	73		
52	34			116	74		
53	35			117	75		
54	36			118	76		
55	37			119	77		
56	38			120	78		
57	39			121	79		
58	3A			122	7A		
59	3B			123	7B		

Decimal	Hexadecimal	Command	Ref.	Decimal	Hexadecimal	Command	Ref.
60	3C			124	7C		
61	3D			125	7D		
62	3E			126	7E		
63	3F			127	7F		

- For #13 `transpose,` subtract 64 from the value.
- For #16 – 19, set values in the local variable 0 – 3, respectively.
- For #28 and 29 `sweep_pitch,` subtract 64 from the value (#29 `sweep_pitch` is multiplied by 24 after this).
- For #65 `porta_on` / `porta_off`, `porta_on` when the value is greater than 64 and `porta_off` when less than 64.
- #120 - 127 are channel mode messages and not control changes.

7.3 Compatibility with SMF Created for MusicPlayer2000

This section lists some differences with SMF created for an AGB sound development tool, MusicPlayer2000 (or MP2000). Note that this list is incomplete.

7.3.1 Track Number

With MP2000, track numbers are assigned in order, starting from the lowest track number. With NITRO-Composer, tracks are numbered using the numbers in the sequence data. Missing numbers are not automatically filled.

7.3.2 Modulation

The modulation value is interpreted differently from MP2000, so adjustments in the value are necessary.

7.3.3 Tuning

MIDI Control Change 24 no longer has the tuning feature.

7.3.4 Simulated Echo

MIDI Control Change 30 and 31 no longer have the simulated echo feature.

7.3.5 Priority

The MIDI control code for priority has been changed from 33 to 14.

7.3.6   Loops on Individual Tracks

With MP2000, the loop starting point is MIDI control code 30 and the value is 100. The loop ending point is MIDI control code 30 and the value is 101.

With NITRO-Composer, the loop starting point is MIDI control code 89 and the value is 0. The loop ending point is MIDI control code 90 (arbitrary value).

7.3.7   Volume Calculation

The method to calculate volume with NITRO-Composer is slightly different than the method used with MP2000, so volume adjustments may be needed.

7.3.8   Tempo when Creating Sound Effects

With MP2000, a tempo of 150 synchronizes perfectly with process base time, whereas with NITRO-Composer a tempo of 240 or 120 synchronizes with process base time.

7.3.9   Quarter Note Resolution

With MP2000, the resolution for quarter notes is 24, whereas with NITRO-Composer it is 48.