Sappy 2006: OMFG HELP!

MFG HELP!

When I try to play a certain song, I get a generic error message. What's up with that?

The `EOT` command is not fully supported. A single note works fine, but full chords fail. The actual error that's *supposed* to appear is "Subscript out of Range", which doesn't help much either.

I'm afraid there's little to be done here but to avoid playing these songs.

Developers, if you have a song with passages like this:

```
.byte EOT, As2
.byte      Cs3
.byte      Fn3
```

...that wouldn't work. `TIE`s start fine, but `EOT`s spanning more than one note break things.

Lacking .S files, Pokémon's "Caves and Darkness" song is a bit difficult to fix but you could look, in the song data, for `0xCE` and `0xCF` which are `EOT` and `TIE` respectively.

My Sappy-enabled homebrew doesn't work in Sappy. The header pointers are all wrong!

Since Classic VB only supports signed Longs/DWords, certain pointer ranges translate to a negative decimal number. I had this problem myself while working on the Demo Jukebox. I fixed this by moving all `SOUND_FILES` to the start of the rom until the pointer was in a correctly interpreted range:

```
$(TARGET_ELF): $(SOUND_FILES) $(.OFILES) Makefile $(DEPENDFILE)
        @echo > $(MAPFILE)
        $(CC) -g -o $@ $(SOUND_FILES) $(.OFILES) -Wl,$(LDFLAGS)
```

This is caused by small programs and the music data being put at the end of the rom by default.

When using the MIDI driver, the instruments are all wrong!

That's because the games don't need to adhere to the General MIDI standard. Any given instrument or "patch" as it's often called can have any sound. It's up to the sound and music artists of the game in

question how much the game patches match up to the General MIDI standard. You can remap the patches with the `<midimap>` XML tag. Please refer to the Sappy.XML documentation for details.

I have those media keys on my keyboard, but some don't work in Sappy like "Eject"!

As of version 1.1, Sappy only listens to Play/Pause (which doesn't pause), Stop, Next, Previous, Volume Up and Volume Down. It's ignoring of Eject is quite natural. Each key must be coded in seperately.

I'm running on a Japanese system, and the text is all garbled!

It's not easy to get Japanese text rendered properly. We're still figuring this out. In the meantime, you can get a non-localized version where you got this.

Built on Wednesday, April 12th, 2006

Sappy 2006: What's New

hat's New

From 1.1 to 1.2

- Super-fast sample loading - ask for details if you dare!
- Anti-earbleeding system to disable unneeded but ear-piercing sounds like some in Final Fantasy 4.
- MIDI support like in the old Sappy.
- MIDI mapping editor.
- MIDI export, thanks to Drag for fixing it.
- Ability to choose which MIDIOUT device to use.
- Various bugfixes.

From 1.0 to 1.1

- Idiot-proof exporters. You'll be notified when you misread "file pattern" for "file name" or "path name".
- Non-braindead assembler. It no longer fails on pointers.
- Volume control slider. Responds to keyboard (if it has focus) and mouse wheel.
- WM_APPCOMMAND support. If you have dedicated media control buttons on your keyboard, Sappy will listen to them.
- Sound loading on byte arrays instead of character strings, as urged by Bouché.
- Feedback while loading a song to play.
- Rudimentary Sappy Classic-style MIDI support with equally rudimentary GameBoy Classic mode and instrument remap.

Wishlist

- Support for more MIDI controls.
- Tempo fix for exports.
- GameBoy Classic mode for the new driver.

- Full sound loading fix. As of 1.1, only PCM waves are properly loaded. Tone generators are still loaded in strings.

Built on Friday, December 22nd, 2006

Sappy 2006: XML definition

ML definition

- Root element SAPPY → one or more ROM elements.
  - Element ROM → one or more PLAYLIST elements, one optional BLEEDINGEARS element, one optional MIDIMAP element.
    - Attribute CODE, required.
    - Attribute NAME, required.
    - Attribute SONGTABLE, pointer, required.
    - Attribute SCREENSHOT, file.
    - Attribute CREATOR.
    - Attribute TAGGER.
    - Element PLAYLIST → one or more SONG elements.
      - Attribute NAME, required.
      - Attribute STEAL, another CODE.
        - Element SONG → the song's name.
          - Attribute TRACK, required.
    - Element BLEEDINGEARS → one or more INST elements.
      - Element INST.
        - Attribute ID, MIDI patch number, or...
        - Attributes FROM and TO, range of MIDI patch numbers.
    - Element MIDIMAP → one or more INST elements, one or more DRUM elements.
      - Element INST.
        - Attribute FROM, MIDI patch number, required.
        - Attribute TO, MIDI patch number, required.
        - Attribute TRANSPOSE.
        - Attribute SECOND. Not used in this version.
        - Attribute THIRD. Not used in this version.
      - Element DRUM.
        - Attribute FROM, note, required.
        - Attribute TO, note, required.
        - Attribute KIT. Not used in this version.

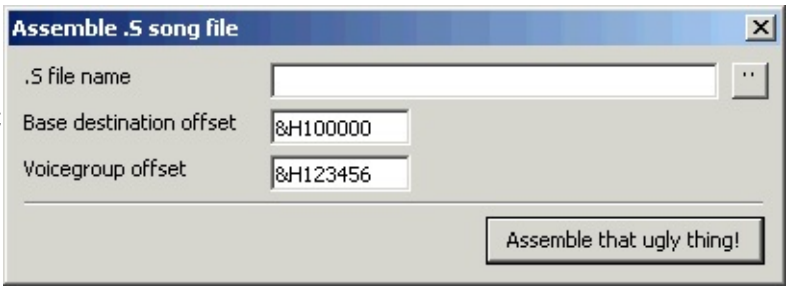
Built on Wednesday, July 26th, 2006

Sappy 2006: Hacker Stuff

ID2AGB and the Built-In Assembler

If you have the MID2AGB converter, you can use it to convert most MIDI files to the Sappy engine's native format. But this is only half the procedure. The resulting .S files must be converted to binary code. This can be done with most standards-compliant assemblers, such as the one used in GCC.

Unfortunately, when using an assembler the pointers are all off and the Voice Group is undefined. We used to hand-edit the .S files in NotePad or whatever and add the intended location in ROM to all referenced pointers.



Sappy 2006 packs a built-in assembler-like tool that's custom-built for the specific purpose of converting Sappy .S files. It doesn't generate a single byte of code, only song data. But it does automatically fix the pointers, and even recognizes the header bytes when it sees them so the new song header can automatically be linked into the Song Table!

age

First, you must copy or rename the file `musicplaydef.s` to `mplaydef.s`. For some reason, Nintendo gave it another name than is referenced in the song files, and copies to `\src` with the proper name. Make sure that `mplaydef.s` is in the same folder as your song files.

After converting your song, picking a voicegroup offset from another song and determining where in the ROM you want to put the new one, open the assembler and fill in all three fields. The rest should be pretty much automatic.

The assembler works only because Sappy song files have certain characteristics:

- No forward jumps, only backwards.

- Words are used exclusively in jumps and the header.
- The only `.global` is also the song name and therefore header label.

Built on Tuesday, November 8th, 2005

Sappy 2006: Hacker Stuff

ong Transplants

Any game that uses the M4A engine to play back its music and sounds can be used in a song transplantation. This used to be a very big task, finding a song's header and hand-dumping each track. Then, you'd have to go through each track and edit every single pointer. One slip-up and the song wouldn't work.

Now all you have to do between exporting and importing the tracks is to edit the tracks' VOICE commands (most tracks having only one of those), and picking a nice voice group that more or less suits your needs.

The following things must be done for a proper song transplant:

- Export tracks
- Edit voice commands
- Import tracks

Built on Friday, November 4th, 2005

Sappy 2006: Hacker Stuff

‹port Tracks

The Export Tracks dialog has a list of track pointers and a pattern-sensitive filename textbox.

If you enter, for example. "`RCR Intro track $T.bin`", you'll get track files called "`RCR Intro track 0.bin`", "`RCR Intro track 1.bin`" etcetera. Any pointers found therein will be automatically recalculated. Entering "`$P`" adds the tracks' pointers to their filenames.

Built on Monday, November 28th, 2005

Sappy 2006: Hacker Stuff

dit voice commands

Because instruments hardly ever match up between games, transplanted songs must be edited to use the target game's instruments. There are two steps to do this:

- Figure out which instruments to use.
- Edit track data.

To do step 1, open the target game and find a song that has the required instruments, or something close. Then press *Extract Samples* and make sure you use the `$I` file name pattern. Identify the instruments you want and write them down somewhere. This is only one of several methods.

To do step 2, open each track file in a hex editor and find the `BD` bytes. They can often be found near the beginning of each file. The byte that follows it is the track's instrument. Replace these values with the ones you wrote down in step 1.

Built on Friday, November 4th, 2005

Sappy 2006: Hacker Stuff

ıport Tracks

The Import Tracks dialog is a little more complex than the Export dialog. Select the track files you want to import in the list box, and enter three pointer values. Defaults are provided, these are the currently selected song's pointers.

Tracks are handled in the order they appear in the list box. To import a later-appearing file before an earlier-appearing file, rename them in Explorer to change their sorting order. In-track pointers are recalculated when needed.

When importing is done, Sappy will ask for confirmation to automatically change the Master Song Table to reflect your changes.

Built on Friday, November 28th, 2005

Sappy 2006: Hacker Stuff

ata formats

mples

Type	Name	Description
u16	type	Not used
u16	stat	0x0000 for oneshot, 0x4000 for forward loop
u32	freq	"Normal" frequency << 10
u32	loop	Loop start position in samples
u32	size	Total number of samples
s8	data[]	Size+1 samples in total. Last byte is zero for one-shots, same as loop pointer for looped samples.

usic headers

Type	Name	Description
u8	NumTrks	Number of tracks in the song
u8	NumBlks	Number of blocks (?)
u8	Priority	
u8	Reverb	
u32	Voicegroup	Pointer to instrument bank
u32...	Track	Pointers to tracks

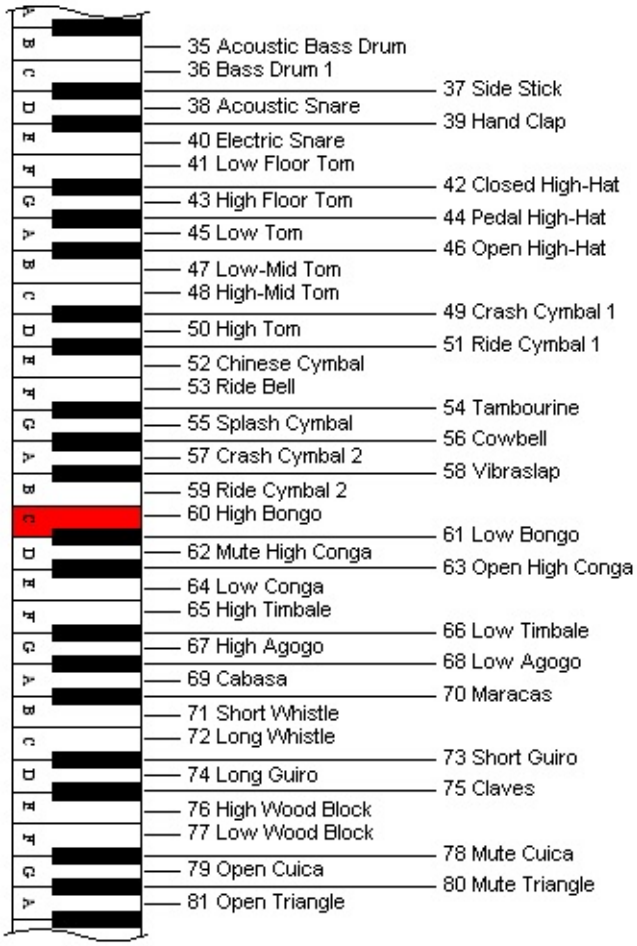
Built on Monday, March 7th, 2006

Sappy 2006: Hacker Stuff

IDI Reference

Note that no single GBA game has to conform to this standard.

um Keys



Nuff said.

tch Assignments

Piano	Chromatic Percussion	Organ
0 Acoustic grand piano	8 Celesta	16 Hammond organ
1 Bright acoustic piano	9 Glockenspiel	17 Percussive organ
2 Electric grand piano	10 Music box	18 Rock organ
3 Honky-tonk piano	11 Vibraphone	19 Church organ
4 Rhodes piano	12 Marimba	20 Reed organ
5 Chorused piano	13 Xylophone	21 Accordion
6 Harpsichord	14 Tubular bells	22 Harmonica
7 Clavinet	15 Dulcimer	23 Tango accordion
Guitar	**Bass**	**Strings**
24 Acoustic guitar (nylon)	32 Acoustic bass	40 Violin
25 Acoustic guitar (steel)	33 Electric bass (finger)	41 Viola
26 Electric guitar (jazz)	34 Electric bass (pick)	42 Cello
27 Electric guitar (clean)	35 Fretless bass	43 Contrabass
28 Electric guitar (muted)	36 Slap bass 1	44 Tremolo strings
29 Overdriven guitar	37 Slap bass 2	45 Pizzicato strings
30 Distortion guitar	38 Synth bass 1	46 Orchestral harp
31 Guitar harmonics	39 Synth bass 2	47 Timpani
Ensemble	**Brass**	**Reed**
48 String ensemble 1	56 Trumpet	64 Soprano sax
49 String ensemble 2	57 Trombone	65 Alto sax
50 Synth. strings 1	58 Tuba	66 Tenor sax
51 Synth. strings 2	59 Muted trumpet	67 Baritone sax
52 Choir Aahs	60 French horn	68 Oboe
53 Voice Oohs	61 Brass section	69 English horn
54 Synth voice	62 Synth. brass 1	70 Bassoon
55 Orchestra hit	63 Synth. brass 2	71 Clarinet
Pipe	**Synth Lead**	**Synth Pad**
72 Piccolo	80 Lead 1 (square)	88 Pad 1 (new age)
73 Flute	81 Lead 2 (sawtooth)	89 Pad 2 (warm)

74 Recorder
75 Pan flute
76 Bottle blow
77 Shakuhachi
78 Whistle
79 Ocarina

82 Lead 3 (calliope lead)
83 Lead 4 (chiff lead)
84 Lead 5 (charang)
85 Lead 6 (voice)
86 Lead 7 (fifths)
87 Lead 8 (brass + lead)

90 Pad 3 (polysynth)
91 Pad 4 (choir)
92 Pad 5 (bowed)
93 Pad 6 (metallic)
94 Pad 7 (halo)
95 Pad 8 (sweep)

Sound Effects

120 Guitar fret noise
121 Breath noise
122 Seashore
123 Bird tweet
124 Telephone ring
125 Helicopter
126 Applause

Built on Wednesday, April 12th, 2006

Sappy 2006: Jukebox

ıkebox Demo Rom

The Jukebox Demo Rom is written by Kyoufu Kawa, based on code from *Catnip Dreams*. Even though it demonstrates simple palette rotation and less simple Variable Width Fonts, it's main focus is to play songs using the M4A engine.

ɔntrols

Left/Right - Select song. Your selection wraps around if needed.
A button - Play the currently selected song.
B button - Stop playing.
Shoulder buttons - Play sound effects.

scellaneous information

The rom's internal name and gamecode is `KAWAJUKE 2K6 - KWJ6`. It has a Maker Code of `0xFF`. It was made with the official Nintendo-brand GBA development kit. It has the unique quality of having all the song data at the *start* of the ROM.

Built on Thursday, September 28th, 2006

Sappy 2006: Jukebox

ıkebox Song Credits

Listed in track order. Instrumentation has been altered and some tracks have been completely removed.

Test song Hand-coded by Kyoufu Kawa

Keitaro's Theme - "Memories"
From "Catnip Dreams", 2005 The Helmeted Rodent.
Composed by Majin Bluedragon.
The game was never released, and songs may return in later games from the same authors.

Monkey Island 2 Theme Song
From "Monkey Island 2", LucasArts.
Recorded from DOSBox.

GRNFINAL
From "The Incredible Toon Machine", Sierra.
Game used MIDI files.

Canon
By Pachelbel.
Found by Google.

Grabbag
From "Duke Nukem 3D", 3D Realms.
Game uses MIDI files.

At Doom's Gate, **Kitchen Ace** and **I Sawed the Demons**
From "Doom", iD software.
Downloaded from DoomWorld.

Eyecatcher
From "Azumanga Daioh - The Anime".

TIM1
From "Lemmings", Psychnosis.

???
From "Ninja Gaiden".

Kawa's Comin'

Original by DJ Bouché, extended version by Baro.

Zero Wing Medley
From "Zero Wing", Toaplan.
Sequenced and arranged by Mars Jenkar.
Downloaded from VGMusic.

Hu-Ha Dschinghis Kahn
By Dschinghis Kahn.
Downloaded from some Russian site.

500 Miles (I'm Gonna Be)
By The Proclaimers.
Downloaded from a karaoke site.

Pallet Town
Ported from Pokémon.

Thexder theme
Sequenced by Jan van Valburg.
Downloaded from VGMusic.

Built on Friday, December 22nd, 2006

Sappy 2006: Registry keys

egistry keys

AutoAdvance
Doesn't do anything.

Bar X state
Collapsed/Expanded state of the ExplorerBar bars.

Driver
Which driver to use.

Force Nice Bar
Use system style or fake it out. Non-XP systems are faked by default.

FMOD Volume
Volume at which to play the music when using the FMOD driver.

Hide Bar
Hide the ExplorerBar completely. Makes the player look like Bouché's original. Tasks can be selected from the menu, but ROM information is unavailable.

Incessant Sound Override
Don't play any Incessant Sounds if true.

Last ROM
Full file name of the last ROM you opened.

MIDI Device
Index of the MIDIOUT device to use.

MIDI in GB Mode
Replaces all instruments with square and saw-waves if set.

mIRC Now Playing
Write Now Playing information to sappy.stt. Use sappy.mrc to show this during IRC sessions.

MSN Now Playing
Send Now Playing information to MSN Messenger.

Nice Menus
When enabled, use Office 2003 style white menus. When disabled,

use older style gray menus.

Reload ROM
Reload the file specified in "Last ROM" when starting up.

Seek by Playlist
When enabled, the Previous/Next buttons go by playlist entries instead of raw song numbers.

Settings Page
The index of the current page on the Settings window. Quick re-entry.

Skin
Skin # to use.

Skin Hue/Saturation
Colorizing values for the skin

Song Repeats
Determines the number of times to repeat playback.

Window Height
Height of the main window in twips.

Window Font (Size)
Allows you to change the font used in the windows. Defaults to "Lucida Sans Unicode", 8 points.

XML File
XML file to use.

Built on Thursday, September 28th, 2006

Sappy 2006: Track byte code

ack byte code

Wxx (0x80++)
: Wait for the specified number of clock ticks

FINE (0xB1)
: Musical term. Ends the track.

GOTO, label (0xB2)
: Unconditional jump.

PATT, label (0xB3)
: Unconditional jump, but remembers where to return. Can be nested up to three times.

PEND (0xB4)
: Ends a pattern and returns to it's call source. Ignored if not called from somewhere.

REPT, xx, label (0xB5)
: Jumps to the specified label xx times. If xx is zero, it repeats ad infinitum like a GOTO. REPT commands cannot be nested, and cannot be specified in a MIDI sequencer.

PRIO, xx (0xBA)
: Sets the priority of the track. xx is from 0 to 255 when directly editing the .s file, 0 to 127 in a MIDI sequencer.

TEMPO, xx/2 (0xBB)
: This sets the tempo of the entire song. xx is from 22 to 510.

KEYSH, xx (0xBC)
: This modulates the key of the track. xx is from -128 to 127. This command cannot be specified in a MIDI sequencer.

The following commands are in "Running Status" mode. This allows operation from their parameters alone, which is good when a whole lot are seen in sequence, like a pitch bend or volume slide.

VOICE, xx (0xBD)
: Selects a patch for this track. xx is from 0 to 127.

VOL, xx (0xBC)

Sets the track volume. xx is from 0 to 127.

PAN, xx (0xBF)
: Sets the track's stereo pan position. xx is from -64 to 63, but if it's a CGB instrument, hardware restrictions limit you to just "left", "center" and "right".

BEND, cV + xx (0xC0)

BENDR, xx (0xC1)

LFOS, xx (0xC2)

LFODL, xx (0xC3)

MOD, xx (0xC4)

MODT, xx (0xC5)

TUNE, cV + xx (0xC8)

These are note commands. They can be used seamlessly with Running Status commands because of their bytecodes.

Nxx [, key [, vel]] (0xD0++)
: Plays a note. xx is the note length from 00 to 96. 24 is a quarter note, 96 a whole. Key is in the "Cn3" format where the first character is the name of the note ("CDEFGAB"), the second either "n" for a natural or "s" for a sharp and the third the octave. There are also "minus octaves", written like "CnM2" and stuff. Octaves effectively range from M2 to 8. From CnM2 to Gn8 covers all 127 MIDI key numbers. Velocity ranges from 0 to 127, but unlike in MIDI, velocity 0 is *not* synonimous to "note off".

EOT [, key] (0xCE)
: Stops the matching TIE note.

TIE [, key, [, vel]] (0xCF)
: This note, which works just like Nxx, continues to sound until the next matching EOT command.