Fire Emblem Music Hacking Tutorial

This tutorial, created by me, Blazer, will guide you step-by-step on how to take a MIDI and insert it into either Fire Emblem 6 or Fire Emblem 7. I have not made an instrument patch for Fire Emblem 8 so you'll have to either find some other way to hack music or bribe me into making one for FE8. :P

Tools Needed:

Required:

Music List.txt

Your MIDI, i.e. a song file ending in .mid

Midi2AGB/Midi2GBA*

Anvil Studio (or another good MIDI editing program, although Anvil is preferable)

HxD (or another good Hex Editing program)

A Fire Emblem ROM (or another GBA ROM—this tutorial is direct towards Fire Emblem hacking, however)

Blazer's Instrument Patch (found in tutorial)

NUPS

Optionals:

Sappy 1.6

Sappy 2005

*(Same programs, I've seen them under both names, I will reference it as Midi2GBA)

YOU CAN FIND MOST PROGRAMS AT HTTP://WWW.FESHRINE.NET/

Part 1: Background Information

Before trying to insert custom music into Fire Emblem or any other GBA game, you should:

- Know how to manipulate a hex editor and its basic commands (go to, find, opening, saving, copying, pasting, and editing)
- Know how to make patches as well as apply patches
- Use Visual Boy Advance to play ROMs, savestate test, etc.

- Know how to back-up ROMs. Backing up is very important. I will not constantly warn you to back up your ROM, but I will on occasion—it's up to you to do it.

Terms: (note: some definitions may have been simplified or otherwise defined as something else for the sake of making it easier to understand, please don't talk to me about technicality terms, I'm a casual hacker and I don't care for 100% accurate definitions.)

MIDI- a song file that contains all the tracks of a song.

Track- one part to a song, a track contains all the info about what a certain instrument should play. Each track has one instrument and the track has all the notes for it.

Instrument- Digital instruments, an instrument is a sound or set of sounds to play. Acoustic Grand is a type of instrument. The track would tell what sounds of the instrument to play, when, for how long, etc.

General Hex Editing Terms- This includes—offset, hex, byte, word, pointer, little-endian, header, etc.

Pitch- How high or low a sound is.

Octave- What set of pitches to use. A lower octave has lower, deeper sounds, while a higher octave will produce higher pitched sounds.

Volume- The loudness, in this case, the loudness of a track or song.

Part 2: Downloading the Programs

Everything you need (excluding illegal ROMs, which btw, a ROM isn't really a program) can be found at http://www.feshrine.net/hacks.html, simply navigate that site and download what you need.

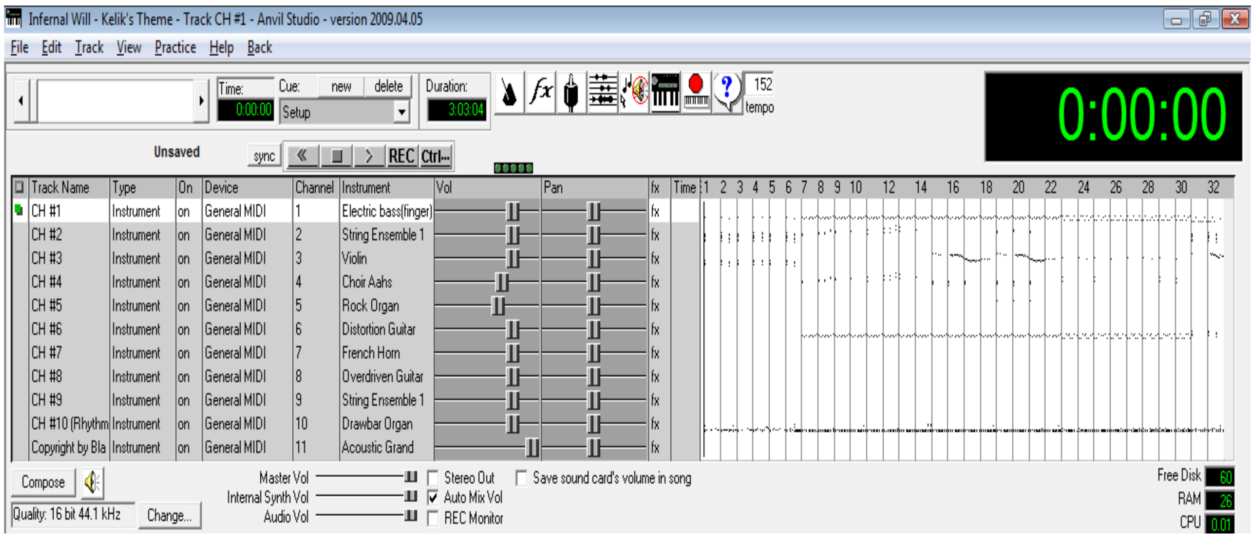
The music list can be found in the documentation section. Anvil Studio and Mid2GBA can be found on the Music Editors page, just as HxD can be found on the hex editors page. The instrument patch can be found in the Patches section and NUPS can be found on the patchers page.

Finding a MIDI is up to you. Use google and http://www.vgmusic.com to find a MIDI of a song you may like or something. Don't ask for help with this.

Part 3: Preparing Your MIDI

Before you insert your MIDI, you need to make sure it is properly prepared. This includes quite a few things. Ease up your mind, it's pretty straightforward and you use a easy program to help you with it.

Load up Anvil Studio, and then load up your MIDI file. In the middle you should see a bunch of tracks.


Infernal Will - Kelik's Theme - Track CH #1 - Anvil Studio - version 2009.04.05

File Edit Track View Practice Help Back

Time: 0:00:00 Cue: new delete Setup Duration: 3:03:04 152 tempo 0:00:00

Unsaved

Track Name	Type	On	Device	Channel	Instrument	fx
CH #1	Instrument	on	General MIDI	1	Electric bass(finger)	fx
CH #2	Instrument	on	General MIDI	2	String Ensemble 1	fx
CH #3	Instrument	on	General MIDI	3	Violin	fx
CH #4	Instrument	on	General MIDI	4	Choir Aahs	fx
CH #5	Instrument	on	General MIDI	5	Rock Organ	fx
CH #6	Instrument	on	General MIDI	6	Distortion Guitar	fx
CH #7	Instrument	on	General MIDI	7	French Horn	fx
CH #8	Instrument	on	General MIDI	8	Overdriven Guitar	fx
CH #9	Instrument	on	General MIDI	9	String Ensemble 1	fx
CH #10 (Rhythm	Instrument	on	General MIDI	10	Drawbar Organ	fx
Copyright by Bla	Instrument	on	General MIDI	11	Acoustic Grand	fx

Master Vol, Internal Synth Vol, Audio Vol; Stereo Out; Auto Mix Vol ✓; REC Monitor; Save sound card's volume in song

Quality: 16 bit 44.1 kHz

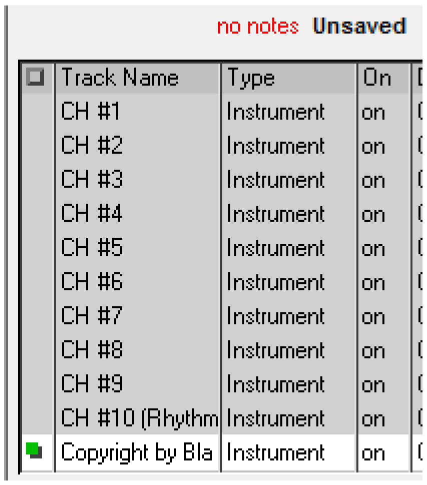
Free Disk 60, RAM 26, CPU 0.01

The middle part is where all your tracks and their information are. To the left is the track name, then the type it is, and the instrument used (example, Acoustic Grand, Violin, String Ensemble 1, etc.).

<u>Step 1: Minimizing Tracks</u>

Having more than 10 tracks in a song will probably screw things up somewhere down the line. I wouldn't even try and do it—I remember a friend telling me he tried to and it failed.

One thing to do is delete any tracks without notes. It will say to the left of the play/pause buttons "no notes" if a track doesn't have any notes. Sometimes there are redundant tracks labeled "Copyright" or something. If there are any, go to Track->Delete at the top of the menu.


no notes Unsaved

Track Name	Type	On
CH #1	Instrument	on
CH #2	Instrument	on
CH #3	Instrument	on
CH #4	Instrument	on
CH #5	Instrument	on
CH #6	Instrument	on
CH #7	Instrument	on
CH #8	Instrument	on
CH #9	Instrument	on
CH #10 (Rhythm	Instrument	on
Copyright by Bla	Instrument	on

That shows a redundant track named "Copyright by Bla Bla Bla" and at the top it says "no notes".

If you still have more than 10 tracks, you should find another MIDI. Sorry but, there are limits in life.

Step 2: Truncating Your Song

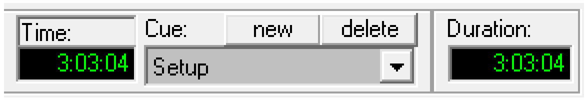
Some songs repeat within themselves. Like a song may be 6 minutes, but at 3 minutes it just repeats itself all over again. Well, in-game, this uses up some space and for maximum efficiency, you'd best get rid of the repeat. Do so by finding out exactly where the song repeats and then going File->Truncate Song-> Delete from Current Position to End.

Going to View-> Composer may help you find the place where it repeats—otherwise just listen to the song and stop once you hear it, then click around in the track area until you get to the point where it repeats. Then do as I say and TRUNCATE!

Step 3: Track Timings

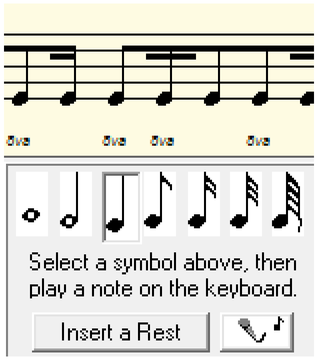
In order for your MIDI to properly repeat in-game, all of the tracks need to be aligned. Otherwise one track might go ahead and play itself before another track has finished its rounds, and then once it repeats in-game, you'll be like "WOAH THE SONG IS ALL MIXE D UP WHAT DO I DO?!!!!".

I got this trick from someone else—I believe it was Charon the Ferryman, a member on my forums. Go to View-> Composer if your composer scroll isn't loaded already. Select the first track, then click in the composer area. Hit "page down" on your keyboard (or if you don't have that, do it slowly by holding the right arrow) until you reach the end of the track. Making sure you are at the end, look at the top.



Time: 3:03:04 Cue: Setup Duration: 3:03:04

Make sure the time and the duration are the exact same. If it isn't, then add rests by clicking the "Insert a Rest" button until it is.



There's the button if you can't find it. Now if your song's duration CHANGES, that's also bad. Press the backspace button to delete rests. Usually one backspace will get it to the perfect timing. In my case, the duration went to 3:03:07 due to an extra rest, and then I pressed backspace once, and now it is back to 3:03:04.

Click on the next track (simply click on the row of the track underneath the column named "Track Name" if you're stuck on how to do select a track, and then make sure the track is highlighted) and do that with EVERY SINGLE TRACK. Yeah, the repeating process makes you hate large #'s of tracks. D:

Once that's done, best save your song. You've now prepared your song for insertion. That was arguably the hardest part of the entire process. :P

Part 4: Applying Blazer's Beta Music Insertion/Instrument Patch

So, get my epic patch from my website at http://www.feshrine.net/hacking/patches.html

I probably sound narcissistic for calling it epic, but everyone needs to feel some good self-accomplishment. I'm just having some fun… XP

Right, back to hacking,

Here's the Readme:

README

This is a private patch supplied by Blazer. It is not for use without permission. Credit must be given.

It adds all instruments at offset 0x107d7d0. Data ends at 0x11b6530. The actual instrument map can be referenced by the offset 0x11ae42c when creating songs.

Back-up your ROM before using and be very careful with this. If you have not gotten this patch directly from Fire Blazer than the data will not be repointed properly and it WILL screw up your ROM if used in combination with a MIDI.

Thank you for your time.

As long as you don't post this tutorial anywhere (you can link people to it, but link them to the WEBSITE, DO NOT REPOST IT SOMEWHERE ELSE), then you're free to use it. Also give credit if you use it in your hack. Not only because I indirectly helped you with your hack but I'm curious to see if people actually make use of things like these, and curiosity is just so hard to control.

Now, with that said, if you read the README (that's the point of it), it says some data is inserted at some scary offset. Well, if you happened to have data at that offset, we have a problem—the patch's data has pointers inside of it. TONS of pointers. The pointers are relative of each other. Unless you want to repoint hundreds of pointers, the data has to go where the patch says it is going. In short, make sure there is nothing at that offset.

So you'll need to move out any data you have there—go use a hex editor and check to see if there is any data between those offsets. If there is, it WILL be overwritten, beware. Also, if you have a clean ROM, this WILL expand your ROM, meaning you will now have to start using UPS patches if you haven't already, and if you simply dislike expanding... well too bad. -_-

BEFORE you use NUPS (find it on feshrine.net) to apply that patch, BACK UP YOUR ROM! I make no guarantees that nothing will go wrong and everything works. It's worked for several people but that doesn't mean it won't screw up your ROM. You have been warned.

That being said, go ahead and apply the patch, because you'll never get anywhere without it.

Part 5: Converting Your MIDI

With that done, time to convert your song to GBA format. Crack out MIDI2GBA. Open up that program called "tr.exe". It has an icon with the text 'ELF'. Put your MIDI file into the folder called 'mid'. The 'mid' folder is inside of the MIDI2GBA folder. Make sure it is your ONLY and I mean ONLY Midi there. In fact, make sure it is your only file there. Otherwise things can get confusing later on and my methods won't necessarily work if you try and insert/convert multiple MIDIs at a time.

Once you're sure it's in, press the "??" button in tr.exe (the ELF program). A pop-up window will come up and your song will either be on the left or the right. If it's on the right, the conversion failed. If it's on the left, it's a success.

Now if it's on the right, don't get all pissed at me. There are some reasons why this might happen.

- Errors in the MIDI. Try to repair it using Anvil Studio. In Anvil, go to File-> Repair and hit "No" to any pop-ups, then save and retry.

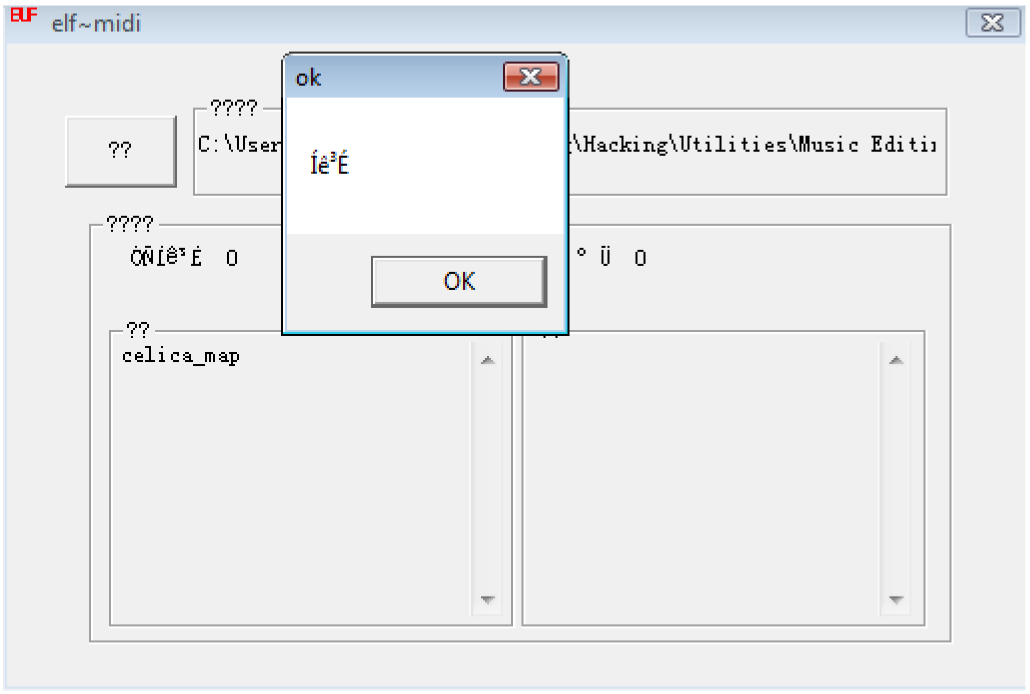
- The following may actually mess it up as well. Don't try to repair the song and then insert, first try to insert and then repair.

- Awkward instruments or something. Perhaps your MIDI file has some instrument that the program "doesn't like". I doubt it, but who knows.

- Bad tracks, some unsupported type of track.

- MIDI file type. Try saving it as MIDI format 1 and midi format 0, although I'm not sure if this makes a difference, maybe it does. It has to be a MIDI by the way—MP3's and WAV's are totally different, don't even try them.
- The file name has some weird symbols in it.
- You could just be unlucky and your song doesn't want to work. This often times happens with Felover3, a hacker who seems to fail at a lot of things… Poor dude.

If you can't get it fixed, I'm sorry, this is a flaw in this method. I can't help you too much besides saying redo the process or try a different MIDI. To be honest, while writing this tutorial I did one myself, but the MIDI I tried to insert ended up being on the right. In fact, so did the 2nd one. Not until I got to the 3rd one did it work like it should.

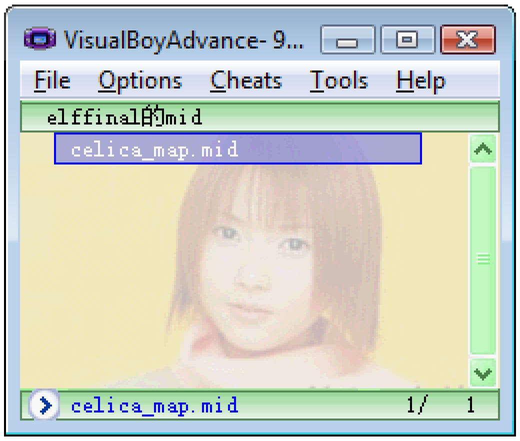


Now, time to test out how your song should sound in-game.

Already?! We already get to test?!

YESZ!

In the MIDI2GBA folder is a ROM called song.gba – load said ROM using VBA and then press "Z" (the equivalent of the "A" button in-game), wait a second, and then listen to your song.



If it's choppy, then that means conversion had some difficulty. If the sounds are a little different or something, then that's due to instruments. Conversion isn't perfect and the GBA hardware has its limitations, thus the song won't sound perfect. If it sounds pretty good, that's great let's continue. If not, try repairing the song, manually fixing it using MIDI editing skills (if you have any), or try another MIDI.

My song, Celica's Map Theme, sounds great, although not exactly like it should, so I am continuing. My other two choices (Awkward Justice from Tales of the Abyss and Rick Roll by Rick Astley) didn't work… it was quite unfortunate really, but such is hacking. :P

Now time to get it to Fire Emblem.

Nub approach: can't we just copy the song from song.gba, the game we just played, into our game?

My approach: Let's just copy the song from the test game into our game!

Hey, something IS simple! Sorta. Let's get down to it.

Part 6: Making Your MIDI Repeat and Transferring it to Your ROM

Sigh, this tutorial is getting really long. And my hands are tired. Yeah. Not cool. Also, I'm making more and more redundant comments like these.

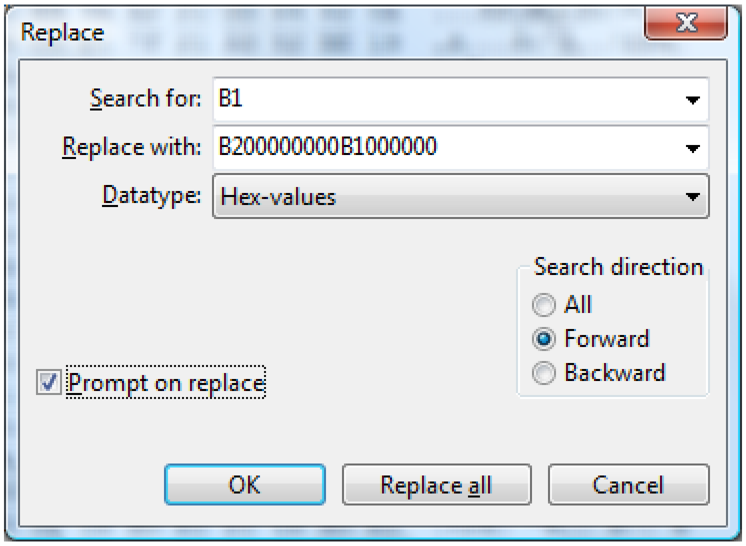
Get your hex editor out, open up your Fire Emblem ROM as well as song.gba. Back-up your ROM, btw.

Go to 0x1B3BB8 in song.gba – this is where your first track starts. Now copy everything from here to the end of the ROM. You've just copied all of the music data as well as the song's header. That's all you need to know for now. :P

Now I'm going to make use of a new method of doing the repeats in the game. What you have to do is paste all that data you just copied into a blank file. Using HxD, just press Ctrl+N or File->New and a new tab will come up. Go to that an press Ctrl+V to paste all of your song data. Bam.

Now here's how this works. The command to end a track is B1. The repeat command is B2 *pointer*. For whatever reason, tr.exe doesn't add a repeat to the tracks, so you have to do it yourself. My easy way is to just use a find and replace feature.

Press Ctrl+R in HxD, or if you're using some other hex editor, find the option to do a find and replace. In the find menu, type in "B1" (hex) and in the replace area type in "B2 00000000 B1 000000"—without the spaces. It should look identical to this.


Search for: B1
Replace with: B200000000B1000000
Datatype: Hex-values
Search direction: [✓ Forward]
Prompt on replace [✓]

With that done, press "OK". If you selected "Prompt on Replace" it will ask whether you want to replace each instance of B1. In most cases you want to replace every one, but SOMETIMES there will be a B1 byte that isn't actually the B1 we want to replace. If that doesn't make any sense, sometimes there will be a ninja B1, but we DON'T want to replace those, because they are ninja. The way to make sure that you don't replace a wrong byte is look at the B1 thing and check to make sure that after the B1 is a "BC" byte. This'll make more sense with a picture.

```
B1 BC 00 BD 04 BE 32 C5 01 BE 22 BF 16 8C D9 58
```

If the B1 and BC are next to each other then I can almost guarantee you want to replace it, so hit "replace" and do that with every instance and once you're finished you're good to go.

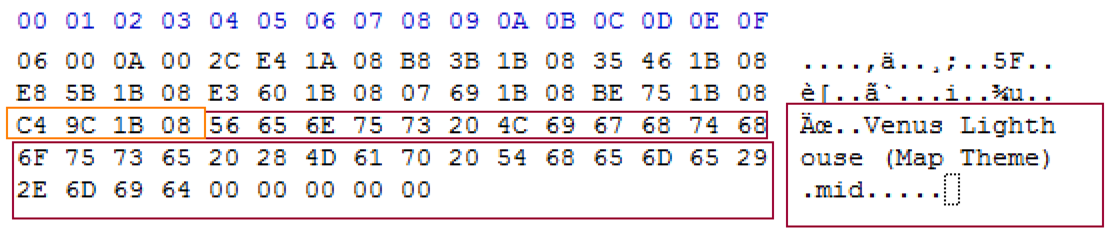
What you've just done is preformatted the song to have repeats. You added in the code, so now you have to insert the MIDI into your ROM and then fix up ALL the pointers. Fun.

As I just said, we need to put the MIDI in your Fire Emblem ROM. Copy all the MIDI data (the one that you just did a find-and-replace function on). We'll need a bunch of free space for this song. I'm going to paste at some offset farther into the game that isn't being used at all. Make sure wherever you paste your edited song/MIDI IS NOT BEING USED and there is plenty of free space around it, just to be safe.

For your info (in case you didn't know), it is best to insert at offsets that end in '00'. Like 0x1200000 is easier to remember than 0x10849C4, although both will work. I just expanded my ROM further and am inserting it at 0x11E4000. That's an easy to remember offset. Because offsets are always important, I am going to write it down. You should be writing down all these offsets in something like a notepad file yourself—do it or you'll suffer consequences later on.

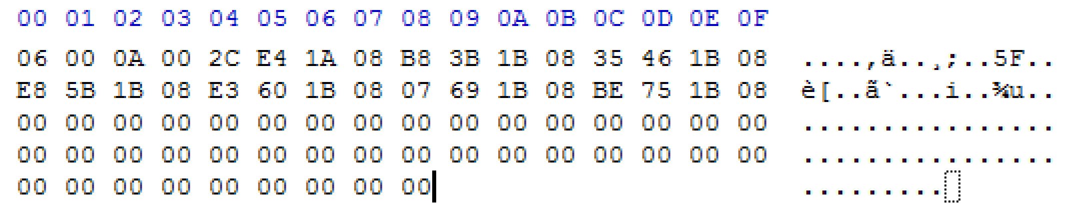
Now, once you paste it, go to the end of wherever your pasted data is… my hex editor (HxD) automatically takes me there. By the way, you should paste write, not paste insert. Paste insert = Ctrl + V and it adds the data in the middle of the game, which messes up pointers. Ctrl + B overwrites the free space you have and doesn't mess up pointers.

Once you're at the end of your song data, you should see to the right (in the ASCII text part) the name of your file, something like "awesome.mid" with "awesome" being the name of your song… If you see that, you are in the right area. Congratulations.


```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

06 00 0A 00 2C E4 1A 08 B8 3B 1B 08 35 46 1B 08   ....,ä..¸;..5F..
E8 5B 1B 08 E3 60 1B 08 07 69 1B 08 BE 75 1B 08   è[..ã`...i..¾u..
C4 9C 1B 08 56 65 6E 75 73 20 4C 69 67 68 74 68   Äœ..Venus Lighth
6F 75 73 65 20 28 4D 61 70 20 54 68 65 6D 65 29   ouse (Map Theme)
2E 6D 69 64 00 00 00 00 00                        .mid.....
```

Boxed in dark red is the text. On the right you can see the name of my MIDI and on the left you can see the hex for that text. 4 bytes before the text is what I consider a garbage pointer. That's all you need to know, it's garbage. Like all garbage, you get rid of it. The name of the MIDI is garbage too, so just '00' all that out. Your end product should be something as beautiful as this.


```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

06 00 0A 00 2C E4 1A 08 B8 3B 1B 08 35 46 1B 08   ....,ä..¸;..5F..
E8 5B 1B 08 E3 60 1B 08 07 69 1B 08 BE 75 1B 08   è[..ã`...i..¾u..
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00                        .........
```

With that, on to the next part!

Part 7: Track Pointers & Repointing

```
06 00 0A 00 2C E4 1A 08 B8 3B 1B 08 35 46 1B 08
E8 5B 1B 08 E3 60 1B 08 07 69 1B 08 BE 75 1B 08
```

This is the final part of your MIDI, which is ironically called the header. The part in the dark red box, the first 4 bytes, are 06 00 0A 00 in this example. 0x06 is the # of tracks I have in my song. My song DOES have 6 tracks.

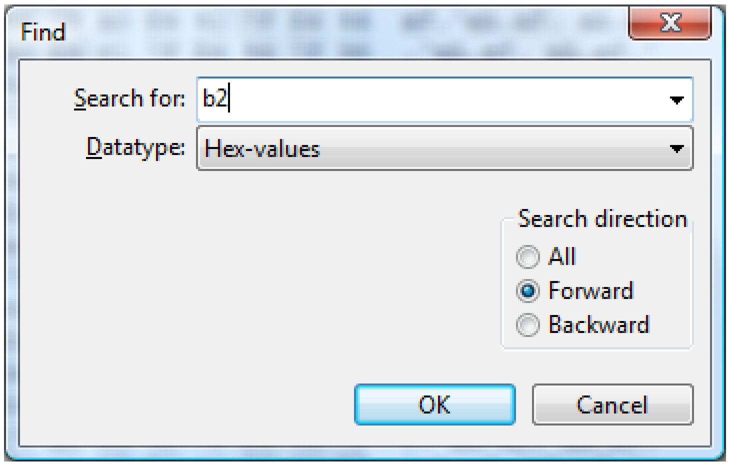
After that is a pointer to the Instrument Maps, also known as Voice Groups, although instrument map is a much better term in my opinion. My patch inserts instrument maps at 0x11AE42C. In “pointer form”, that is 2C E4 1A 09. If you notice, the only difference between the current pointer and what the new pointer should be is the ‘09’, so change that. Once you do, you will have set up your instruments for the

song. Simple as that. Not only this, but any of the songs that are inserted with this method will use the exact same instrument map, so there’s no hassle. :D

After that, in the purple box, are the pointers to the tracks. Right now they still point to our old tracks, eww. A pointer is 4 bytes and if you notice there are 6 sets of 4 bytes up in that data I posted, or 6 pointers. 6 pointers for 6 tracks. Your song will probably be different. It might have 10 pointers or 3 pointers and the pointers won’t all be the same as mine.

Anyway, first we need to add in the repeat pointers. If you remember we added the B2 code to the song for every track earlier on. What we were doing was adding the repeat code, 0xB2, and space for us to put in a pointer. The pointer points back to the beginning of the track so that the track just plays over again… and over again, and over again. :P

Go to the offset where you put your data. For example I put my data at 0x11E4000. From now on, write down offsets, because it is very important to for you to repoint stuff. Do a search for the hex 0xB2 and you’ll get to the end of your track.


Find

Search for: b2

Datatype: Hex-values

Search direction
- All
- [✓ Forward]
- Backward

OK | Cancel

```
B0 B0 B0 B0 B0 B0 B0 B0 A0 B2 00
```

Now the beginning of this track is at 0x11E4000 for me. You just wrote this offset down, it'll be something different for you, depending on where you put your song. In pointer form that is 00 40 1E 09.

About turning offsets into pointers:

Break up the offset into parts:

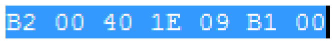
AB + CD + EF + GH = GH + EF + CD + (AB+8)

So 0x00123456 would turn into

56 34 12 08

And 01 1E 40 00 becomes 00 40 1E 01, but you add an '08' to the last byte making it 00 40 1E 09. Bam.

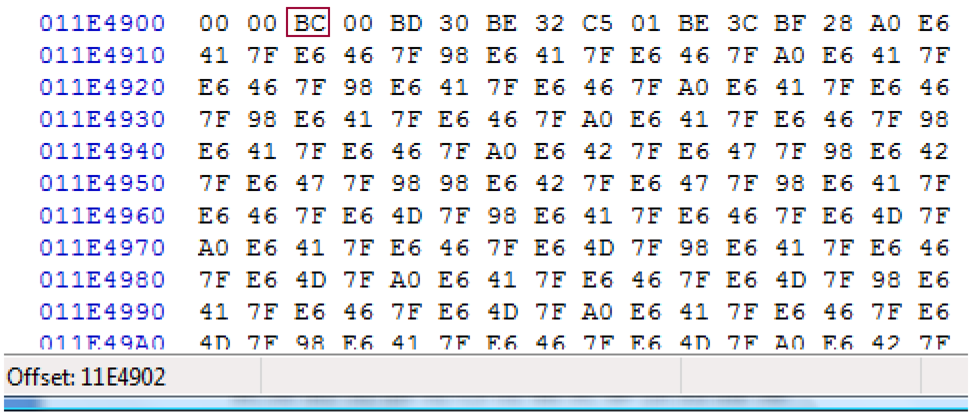
Now type that in where the 00's are between the B2 and B1. Like so.



```
B2 00 40 1E 09 B1 00
```

Now that you've done that, there will be a "BC" byte soon after that. "BC" is the first byte of a track. I could load you with lots of more information, but this is confusing enough as it is, so if you're looking for documentation on how track data is made up, look at the end of this tutorial.

Anyway, we want to know where the beginning of the next track is so that we can tell the game where to repeat to. So find out the offset of the next track—look for the magical BC.



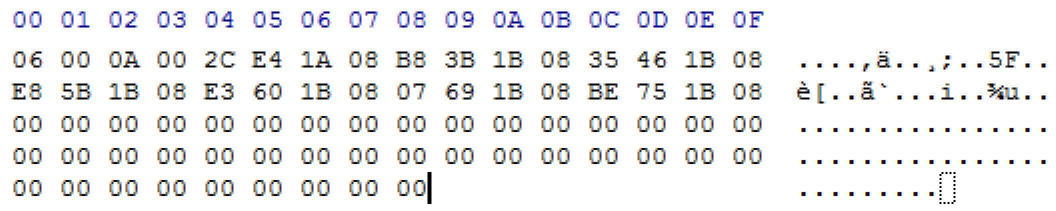
```
011E4900  00 00 BC 00 BD 30 BE 32 C5 01 BE 3C BF 28 A0 E6
011E4910  41 7F E6 46 7F 98 E6 41 7F E6 46 7F A0 E6 41 7F
011E4920  E6 46 7F 98 E6 41 7F E6 46 7F A0 E6 41 7F E6 46
011E4930  7F 98 E6 41 7F E6 46 7F A0 E6 41 7F E6 46 7F 98
011E4940  E6 41 7F E6 46 7F A0 E6 42 7F E6 47 7F 98 E6 42
011E4950  7F E6 47 7F 98 98 E6 42 7F E6 47 7F 98 E6 41 7F
011E4960  E6 46 7F E6 4D 7F 98 E6 41 7F E6 46 7F E6 4D 7F
011E4970  A0 E6 41 7F E6 46 7F E6 4D 7F 98 E6 41 7F E6 46
011E4980  7F E6 4D 7F A0 E6 41 7F E6 46 7F E6 4D 7F 98 E6
011E4990  41 7F E6 46 7F E6 4D 7F A0 E6 41 7F E6 46 7F E6
011E49A0  4D 7F 98 E6 41 7F E6 46 7F E6 4D 7F A0 E6 42 7F
Offset: 11E4902
```

At the bottom left of HxD it tells you the offset of wherever you are. So that byte is at 11E4902. Yes, great. I'm going to write that down, then do a search for B2 again. Then I'm going to take that offset, put it into pointer form (02 49 1E 09) and then place it in the middle of the repeat code. Then I'm going to find the next BC and start the process ALL over again, writing down the offsets each time, until I get to the final repeat code.

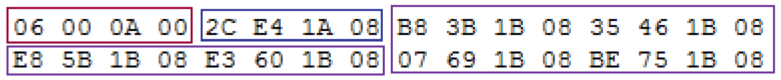
And, stinks for you, but you're going to have to do this too, and the more tracks you have, the more frustrating and tedious it gets.

Eventually you should see your track header again, you know, the 06 00 0A 00 bytes, where “06” is the number of tracks in your song. Once you see that, you know you’ve finished doing the repeating for all the tracks. Congratulations, now you have to repoint the entire track header.

Take all those offsets of the track headers and repoint the tracks.



```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
06 00 0A 00 2C E4 1A 08 B8 3B 1B 08 35 46 1B 08   ....,ä..¸;..5F..
E8 5B 1B 08 E3 60 1B 08 07 69 1B 08 BE 75 1B 08   è[..ã`...i..¾u..
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00 00 00 00 00 00 00 00 00                        .........
```



```
06 00 0A 00 2C E4 1A 08 B8 3B 1B 08 35 46 1B 08
E8 5B 1B 08 E3 60 1B 08 07 69 1B 08 BE 75 1B 08
```

Remember a pointer is 4 bytes. There are 24 bytes there so if each pointer is 4 bytes then that means there are 6 pointers, because my song has 6 tracks. If your song has 10 tracks, that means you have to replace 10 pointers with the 10 offsets of the beginning of each track. Great. Do that and save your game/ROM so you don’t lose all your work. Hahah.

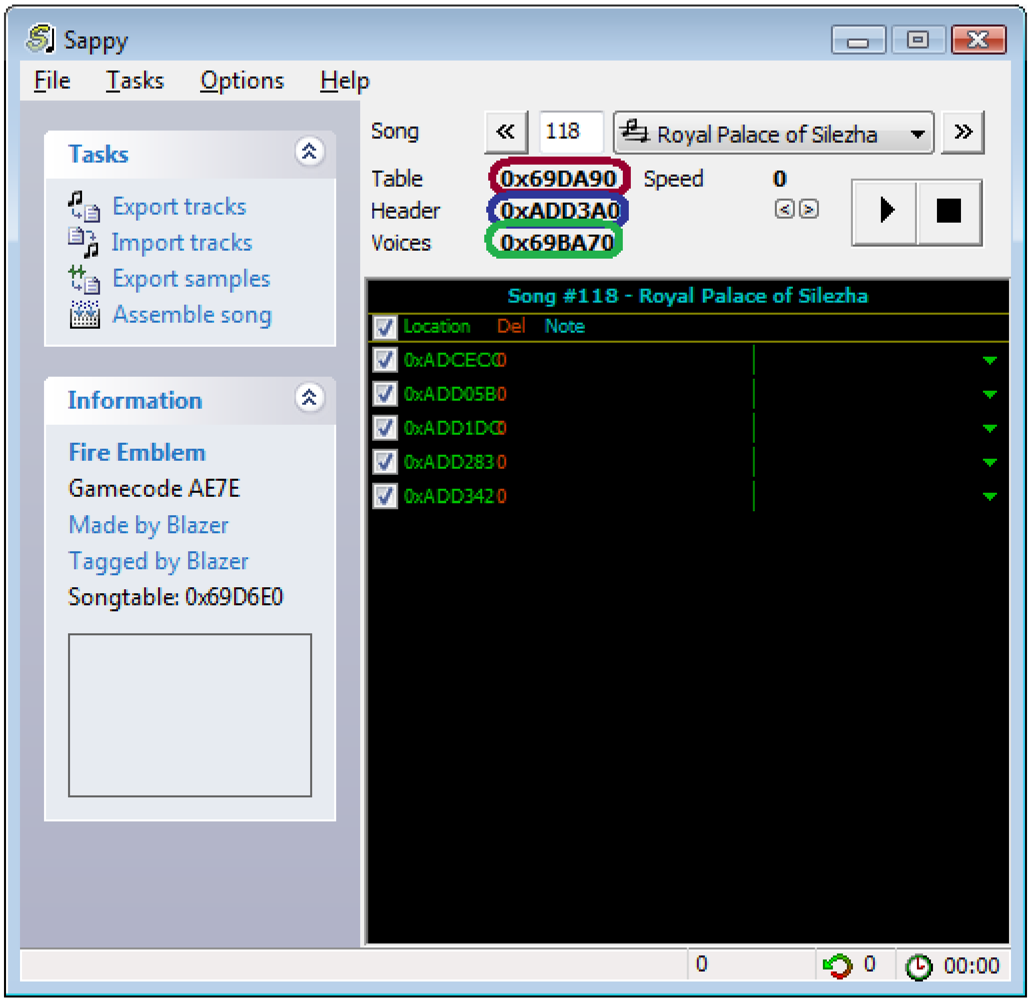
Part 8: Finale- Assembling Your Song & Profit

This differs for each game.

Use the Music List for each game (comes with Nightmare Modules) to find out which song you want to be replace. You can also replace a “blank” song.

Now, take the hex of that song and note it down. I am going to replace song 0x7A in Fire Emblem 7, which is a “Nothing” song. But it’s OK. :D

Now you need to find out where in the pointer table that song is. You can optionally use Sappy 2005 to load up your game, find the song, and then look at Sappy to find the offset of it.



Song 118 Royal Palace of Silezha

Table 0x69DA90 Speed 0

Header 0xADD3A0

Voices 0x69BA70

Song #118 - Royal Palace of Silezha

Location	Del	Note
0xADCEC0		
0xADD05B0		
0xADD1D0		
0xADD2830		
0xADD3420		

Fire Emblem

Gamecode AE7E

Made by Blazer

Tagged by Blazer

Songtable: 0x69D6E0

0 0 00:00

"Table" is where the pointer to the header is. That offset there, circled in red, is what we want.

"Header" (blue) is the offset of the song header. We just made that—it includes pointers to all the tracks, tells the game how many tracks, and where the instrument map is.

"Voices" (green) is the offset of the instrument map. My patch makes a universal version and the offset of it is 0x11AE42C. That's also in the song header, and we've already repointed that too. Just a little bit more!

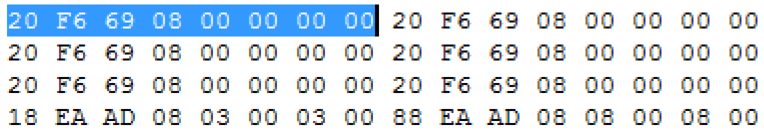
Go to the "Table" offset with Sappy. If you don't have Sappy, follow the directions below

1. For FE7, go to 0x69D6E0. For FE6, go to 0x3994D8. For FE8, go to 0x224470. For any other game, you'll need to use Sappy to find out where the Song Table is.

2. Take your song # (mine is 0x7A—yours is whatever you got from the Nightmare Module's Music List) and multiply it by 8. Use Microsoft Calculator or something (make it Scientific and then select hex—it can multiply hex, that's right). You could probably find some calculator online too. Then take that

number and add it to the song table offset. So for me, 0x7A times 8 = 0x3D0. 0x3D0 + 0x69D6E0 = 0x69DAB0. That is the offset of my song.

Alternatively, for Fire Emblem 7, I made a Music Array Editor which can help you edit any of the songs up to a certain point, but it's limited due to an incomplete music list and it's only for FE7.



```
20 F6 69 08 00 00 00 00 20 F6 69 08 00 00 00 00
20 F6 69 08 00 00 00 00 20 F6 69 08 00 00 00 00
20 F6 69 08 00 00 00 00 20 F6 69 08 00 00 00 00
18 EA AD 08 03 00 03 00 88 EA AD 08 08 00 08 00
```

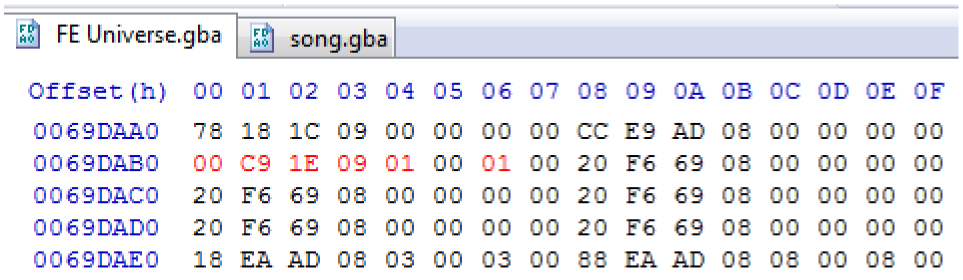
What's highlighted is my song's info. The final bit. The pointer is a pointer to the song's header. Mine is at 0x11EC900. Good luck if you didn't write it down like a good boy.

0x11EC900 = 00 C9 1E 09, so I type that in. The next 4 bytes, which are currently just 00000000, have to do with priority.

00 = Top Priority, 01 = Map Song Priority, and I had others noted down but pretty much the higher the # gets the lower priority it has. Sound effects have a high priority of 6-8 generally, while backgrounds music generally has 0-1, misc. music has 2-3, and yeah… You should be able to figure things out. Most of the time you'll just need to use 00 or 01 anyway.

01 00 01 00 is necessary for map songs to work correctly, or else when a unit goes into battle the map music will stop playing. Just letting you guys know about that.

My final result is:



```
FE Universe.gba   song.gba

Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0069DAA0   78 18 1C 09 00 00 00 00 CC E9 AD 08 00 00 00 00
0069DAB0   00 C9 1E 09 01 00 01 00 20 F6 69 08 00 00 00 00
0069DAC0   20 F6 69 08 00 00 00 00 20 F6 69 08 00 00 00 00
0069DAD0   20 F6 69 08 00 00 00 00 20 F6 69 08 00 00 00 00
0069DAE0   18 EA AD 08 03 00 03 00 88 EA AD 08 08 00 08 00
```

As usual, red is the edited stuff. I save, and then I test. You can test using Sappy v1.6 or just go straight into Visual Boy Advance. Obviously you need to make your song play in game, which means you have to somehow load it, whatever that way is. If you're replacing something like the title theme, then that's really easy to test, because you just load up the game and bam. If you're replacing a song that currently wasn't used, it's up to you to test that song out.

Part 9: Possible Errors & Wrap-UP

Hopefully your song works. If it doesn't, error could be

- Number of tracks
- Song wasn't converted properly (program's fault)
- Faulty pointers (very very likely, double check pointers, check for 08/09 for the last byte)
- Messed up data
- Overwrote some important data
- Edited the wrong pointer (rather than messing it up)
- You inserted some data somewhere, thus shifting everything, making pointers go wrong
- Some sort of pointer error >_>
- The song simply doesn't want to play?
- You aren't playing the right song in-game, you think you're playing the edited song but you really aren't
- I dunno dude. Leave me alone. This tutorial is 4,900+ words and 19 pages. I'M FREAKIN TIRED.

Once you do a couple, you can make up your own tricks to make it go faster, and you'll know how it all works, making things go a LOT faster. I can probably insert a 5 track song in 30 minutes, a 10 track song in like an hour and a half (yeah # of tracks can totally make things harder…). And I haven't done a ton of insertions either. A little bit of practice and you'll make every minute of music insertion worth it.

Update: I recently inserted like, a 6 track song in 10 minutes, so it just takes some practice and then things like repointing and knowing what to do will come REALLY easy. ^_^

Part 10: Documentation and Stuff

Commands:

BC PP

BD II

BE VV

These are commands that set stuff. There are other commands too but I won't get into it. The first is BC PP where PP = octave. 0x00 is default while adding or subtracting 0x0C will get you lower and higher octaves, for example, 0x0C or 0xF4.

BD II sets the instrument. The conversion already does this for you but if you want to change it you can do so manually. The instrument is the one labeled in Anvil Studio—[you] take that # and then subtract one, then convert the number from decimal to hex. Wala, you have your instrument.

BE VV sets the volume, IIRC. VV is a # between 0-127, except in hex.

You shouldn't need to change this but if you want to, it's available to you, and it's better than having to edit the MIDI with anvil studio and then reinsert everything, IMO, assuming you don't have to make massive changes to the tracks.

B0 = Rest

B1 = End Son

B2 = Repeat

Some more commands for you to know*.

*(I am calling them commands, even if the name sometimes doesn't make sense)

Atrius' Notes:

The audio track data is a list of commands and arguments for playing music/sound effects.

Anything below 0x80 is an argument, and anything 0x80 and above is a command.

0x80 - 0xB0 = Rest commands of varying lengths, to wait a certain amount of time before playing each note.

0xB1 = Stop command, to signal the end of the audio track. From what I've noticed even if a song loops it has one of these after the loop command.

0xB2 = Jump command, to jump the playback pointer to another location in memory. Often used to loop songs by jumping back to the beginning of it.

0xB3 = Jump command where playback can jump back to after it again later. Used when you want to repeat part of a song before continuing with the rest of it.

0xB4 = Return to last 0xB3 command.

0xB5-0xBA = Unidentified

0xBB = Set Tempo

0xBC = Set Pitch offset

0xBD = Set instrument

0xBE = Set Volume

0xBF = Set Panning

0xC0 - 0xCE = Unidentified

0xCF - 0xFF = Play notes of varying lengths.

So to break apart the track information I gave you:

BE FF BC 00 BB 01 BD 00-CF 3C 7F B0 B1

BE FF = A volume command with an argument of 0xFF (Volume is an odd exception to the rule that arguments must be less than 0x80)

BC 00 = Set the pitch offset to zero

BB 01 = Very low tempo so that the note is drawn out long enough for voice samples to play their full length.

BD 00 = Set the instrument to the first entry in the instrument map.

CF 3C 7F = Play a note with a pitch of 0x3C (Right in the middle of the available range), and a volume of 0x7F (The maximum available)

B0 = Wait long enough for the note to finish playing

B1 = End the audio track

Charon's Notes:

Pitch

Pitch is very easy to understand. As Atrius noted, it is follows the byte 0xBC (which is usually the first byte in a given track anyhow). Pitch works like this:

0x00 is the base, equivalent to Middle C.

0x0C is one octave higher; High C.

0xF4 is one octave lower; Low C.

If you haven't noticed, it goes by increments of 0x0C, so for higher increments, they would be 0x18, 0x24, ect, while for lower increments they would be 0xE8, 0xDC, ect.

Tracks can be transposed by any value, however, so if you wanted it to be in a different key, you could change these values. If done, make sure to add/subtract 0x0C when raising/lowering octaves.

Tempo
Tempo is simply the tempo transfered into Hex. So, if your original tempo was 80, you would put in 0x50. It follows a 0xBB byte.

Instruments
The instruments correspond with the instruments of the voicegroup. Although the instruments can be anything you want, the typical way is to have the instruments correspond directly. It follows a 0xBD The direct map:

PIANO
0x00 Acoustic Grand (often used as a drum kit)
0x01 Bright Acoustic
0x02 Electric Grand
0x03 Honky-Tonk
0x04 Electric Piano 1
0x05 Electric Piano 2
0x06 Harpsichord
0x07 Clav

CHROMATIC PERCUSSION
0x08 Celesta
0x09 Glockenspiel
0x0A Music Box
0x0B Vibraphone
0x0C Marimba
0x0D Xylophone
0x0E Tubular Bells
0x0F Dulcimer

ORGAN
0x10 Drawbar Organ
0x11 Percussive Organ
0x12 Rock Organ
0x13 Church Organ

0x14 Reed Organ
0x15 Accoridan
0x16 Harmonica
0x17 Tango Accordian

GUITAR
0x18 Acoustic Guitar(nylon)
0x19 Acoustic Guitar(steel)
0x1A Electric Guitar(jazz)
0x1B Electric Guitar(clean)
0x1C Electric Guitar(muted)
0x1D Overdriven Guitar
0x1E Distortion Guitar
0x1F Guitar Harmonics

BASS
0x20 Acoustic Bass
0x21 Electric Bass(finger)
0x22 Electric Bass(pick)
0x23 Fretless Bass
0x24 Slap Bass 1
0x25 Slap Bass 2
0x26 Synth Bass 1
0x27 Synth Bass 2

STRINGS
0x28 Violin
0x29 Viola
0x2A Cello
0x2B Contrabass
0x2C Tremolo Strings
0x2D Pizzicato Strings
0x2E Orchestral Strings
0x2F Timpani

ENSEMBLE
0x30 String Ensemble 1
0x31 String Ensemble 2
0x32 SynthStrings 1
0x33 SynthStrings 2
0x34 Choir Aahs
0x35 Voice Oohs
0x36 Synth Voice
0x37 Orchestra Hit

BRASS

0x38 Trumpet
0x39 Trombone
0x3A Tuba
0x3B Muted Trumpet
0x3C French Horn
0x3D Brass Section
0x3E SynthBrass 1
0x3F SynthBrass 2

REED
0x40 Soprano Sax
0x41 Alto Sax
0x42 Tenor Sax
0x43 Baritone Sax
0x44 Oboe
0x45 English Horn
0x46 Bassoon
0x47 Clarinet

PIPE
0x48 Piccolo
0x49 Flute
0x4A Recorder
0x4B Pan Flute
0x4C Blown Bottle
0x4D Shakuhachi
0x4E Whistle
0x4F Ocarina

SYNTH LEAD
0x50 Lead 1 (square)
0x51 Lead 2 (sawtooth)
0x52 Lead 3 (calliope)
0x53 Lead 4 (chiff)
0x54 Lead 5 (charang)
0x55 Lead 6 (voice)
0x56 Lead 7 (fifths)
0x57 Lead 8 (bass+lead)

SYNTH PAD
0x58 Pad 1 (new age)
0x59 Pad 2 (warm)
0x5A Pad 3 (polysynth)
0x5B Pad 4 (choir)
0x5C Pad 5 (bowed)
0x5D Pad 6 (metallic)

0x5E Pad 7 (halo)
0x5F Pad 8 (sweep)

SYNTH EFFECTS
0x60 FX 1 (rain)
0x61 FX 2 (soundtrack)
0x62 FX 3 (crystal)
0x63 FX 4 (atmosphere)
0x64 FX 5 (brightness)
0x65 FX 6 (goblins)
0x66 FX 7 (echoes)
0x67 FX 8 (sci-fi)

ETHNIC
0x68 Sitar
0x69 Banjo
0x6A Shamisen
0x6B Koto
0x6C Kalimba
0x6D Bagpipe
0x6E Fiddle
0x6F Shanai

PERCUSSIVE
0x70 Tinkle Bell
0x71 Agogo
0x72 Steel Drums
0x73 Woodblock
0x74 Taiko Drum
0x75 Melodic Tom
0x76 Synth Drum
0x77 Reverse Cymbal

SOUND EFFECTS
0x78 Guitar Fret Noise
0x79 Breath Noise
0x7A Seashore
0x7B Bird Tweet
0x7C Telephone Ring
0x7D Helicopter
0x7E Applause
0x7F Gunshot (also commonly used as a drum kit)

Volume
Volume is a simple concept as well, but unlike most of the song arguments, it must be a value

less than 0x80, or else (for some whacked out reason) it acts as a rest o.o

It follows 0xBE.

Panning
I need to investigate panning a little more, but I do believe that 0x00 is the neutral value. Either that, or it's 0x0C. I believe that pan's range is from 0x00 to 0x0F; Atrius, correct me if I'm wrong.

Also, about 0xB1 - if your track doesn't end in it, it will cause the game to crash and make that horrible noise some of us know too well o-o

Part 11: The Music Hacking Run-Down (Shorter Version of Tutorial & Walls of Text)

Let's say you've already hacked music before but you forgot how it works. Perhaps you just want to know the general idea of how to hack music or the general layout of GBA game's music, whether that game be Fire Emblem (which is the focus of this tutorial) or not. This is the section for you. It won't go into huge detail but it'll still give directions and whatnot.

First you have to download all the programs related to music hacking. Once you've unzipped them and everything, take your MIDI and load it in Anvil Studio.

What we're going to do here is have the MIDI formatted so it can be properly inserted. To start, we don't want the song to repeat inside of the MIDI, because we're going to have the game repeat the song for us. If the MIDI repeats within itself then we're just consuming space. Use Anvil Studio to truncate the song such that it only plays once.

Next we have to get rid of all unnecessary tracks to minimize tracks. I don't recommend inserting a song with more than 14 tracks. I've done 14 tracks but on occasion I actually have problems with them. It's best to limit a MIDI to 10 tracks or under if you can, just to be safe. Delete any tracks without any notes (Anvil Studio will have some red text at the top left letting you know) and delete any tracks that are already muted. If you still have too many tracks, look for some redundant tracks that only have a few notes or have minor instruments that you can live without.

After optimizing the number of tracks you will need to equalize your track's lengths. Making sure that you go to View-> Composer, select a track, go all the way to the end of it, and make sure that the track length and the song length match exactly. If it doesn't, insert rests until the track length matches the song length. Make sure you don't add so many rests to the track that it makes the song longer. If you make the song longer it'll mess up the other tracks and it'll also be adding rests to the end of the song, so when your song repeats, it will have a pause in between the repeat, and thus sound bad. That being said, optimize the song to be repeated properly with equal track lengths—with all tracks.

You can now save your MIDI (I recommend to use the "Save as" option in Anvil Studio to make a separate copy with the new edited MIDI). Go to where you saved the MIDI and drag it into the 'midi' folder inside of MIDI2GBA, making sure that the MIDI you are going to insert is the ONLY file inside of that folder. Double-click tr.exe in the MIDI2GBA folder and hit the ?? button. Click "OK" and then look in the two columns. If your song is on the left side, the program and MIDI worked fine… well, they should have, anyway. If it's on the right side, something is up with the MIDI, in which case please refer to part 5 of the tutorial.

If you have not already applied the instrument patch for FE6/FE7, please do so. If there is no instrument patch for your game, what you can do is just apply the patch to a clean FE6/FE7 game anyway and copy the new data in the expanded section of the game (0x1000000 to the end of the game) and copy it to whatever game you are hacking. If you don't get what I mean by this, then oh well, this tutorial is optimized for Fire Emblem hacking, but as usual I just keep FE8 out of the mix.

Anyway, the patch aside, load the ROM "song.gba" (located in the MIDI2GBA folder) into Visual Boy Advance or your GBA emulator of choice (VBA is the best GBA emulator though…). Press "A" or your keyboard alternative which will make the ROM play your inserted song. Make sure it plays nicely and whatnot. Then close the ROM.

You've now heard your song play in a GBA ROM, so hopefully that's motivation to continue this music insertion process. Load your ROM (for example, your hacked Fire Emblem 7 ROM) and song.gba in a hex editor. I most definitely recommend using HxD because it has all of the features needed and that's what I use. Copy all the data from 0x1B3BB8 to the end of the game into a new, blank file. Use a find and replace function to find "B1" and replace it with "B2 00 00 00 00 B1 00 00 00". Do that to all instances—you should find as many "B1" bytes as you have tracks. Once you're done with the find and replace function go to the end of the ROM and delete the ASCII text that has the name of your MIDI. Also delete the 4 bytes before the text as they are part of a stupid pointer that you don't need.

Now copy the data in this file you've just done some stuff with and paste write *(for HxD, use Ctrl+B, NOT Ctrl+V, Ctrl+V will INSERT data and possibly mess up your ROM)* your data into a lot of clear, open space in your FE7 ROM. Make sure there is plenty of space just to be safe. You will need enough space for all of the data you are pasting so just look at HxD to find out how much space that exactly is.

Now go to the beginning of where you pasted your data and write that offset down. In fact, write every relevant offset down from now on. You will need it. I recommend just opening a notepad and typing important offsets.

Do a search for 0xB2. You'll find the data B2 00 00 00 00 B1 00 00 00. The first 4 00's are where your pointer will go. They point to the beginning of the track. This data here is the repeat code and for the track to repeat you need to tell it to play itself over from the beginning of the song. So the pointer should go to the beginning of the song, which I told you to write down the offset. Put it into pointer form (little endian, noting that offsets have an '08' at the beginning of them, such that the offset 005C3984 will become 08 5C 39 84 which will finally be 84 39 5C 08) and enter it into that space of 4 00's. Now right after the "B1 00 00 00" should be a "BC" byte. This is the beginning of the NEXT track. Write down the offset that the BC byte/write the offset of the beginning of the track. Do another search for B2 and do the same thing you just did with the pointer. Repeat this for every track until you end up noticing that there is no “BC” byte for the next track because you’ve added the repeat code for every track. After the final “B1 00 00 00” part you’ll end up at the header (TT 00 0A 00 where “TT” is the number of tracks your song has, in hex). After the first 4 bytes is a pointer to the instrument map. My patch inserts it such that all you have to do is turn the ‘08’ in that pointer to a ‘09’. All the other pointers in the song’s header are pointers to the tracks.

Assuming you listened to me and wrote down all the offsets of the tracks, you can now take those offsets and type them in “pointer form”, one after another, until you’ve replaced all the old track pointers with the new track pointers.

Save your ROM and go to your song’s song table. You can use Sappy 2005 to find this or look at the full-length tutorial for specific offsets for Fire Emblem games. Alternatively there is also an FE7 Music Array Editor which lets you edit the pointers and priority of songs in the song table/array. You’ll now need to find which song you’re going to replace.

Let’s say you’re replacing song 0x4C, whatever that is. That’s great. Each entry is 0x08 bytes—at least it is in Fire Emblem games, I’m not sure about other games—so multiple 0x4C by 0x08. Add this to the offset of the music array (for example, FE7’s music array is at 0x69D6E0). Go to that offset and replace the pointer here with a pointer to your new song’s header. Change the priority accordingly noting that 00000000 is for background music, 000100001 is for map music, and lower numbers are for various things, see the full-length tutorial for more details. XP.

You’ve now done a lot of stuff. Let’s think it over… we’ve formatted the song, got it converted to GBA format, added repeats, made an appropriate song header, made sure that the song has appropriate instruments (it’ll use standard MIDI instruments) and told the game where the song is as well as what priority it has. All we have to do now is make the game use that song. That’s up to you—if you’re replacing something like the title theme then that’s not a problem because the song already plays in game. If you’re replacing some random song or an unused song then you’ll have to make the game use that song yourself which is up to you. For Fire Emblem I suggest using the Chapter Data Editor, Sound Room Editor, or Boss Music Editor to get your song to play in-game if it doesn’t already.

Hopefully it sounds nice and well. If you need more help or if the song had some issues, please see the full-length tutorial (parts 1-9). This is mainly to give the general idea without completely explaining anything, dividing into parts, etc.

Part 12: Credits & Thanks

My last part, I would appreciate credits and thanks for this tutorial. I sacrificed a ton of time creating this and learning how music works and stuff. I’m not trying to complain butaAll I ask is that if you use it, you put my name (Blazer) and Charon the Ferryman somewhere obvious, like in the credits. Thanks for your understanding – I hope this guide helped you.

Special Thanks To:

Atrius – Lots of information about music editing, helped me insert WAV's (which is NOT explained here)

Charon the Ferryman – her tutorial on music editing helped and inspired me to make my own tutorial. She also gave me some one-on-one help with how to hack music and I took her method and just expanded on it and stuff like that.