Music Box Programmer's Guide

Version 1.03.00 - 24/05/99 - 26/02/01
(C) Black Box <http://blackbox.resource.cx/> 2001

Table of Contents

Introduction

First, before you start implementing the Music Box music and soundeffect player into your own program, read carefully the Music Box User's Guide. (Several times if it is necessary.) You need a good overview about its possibilities and nothing will be repeated in this booklet what is already written in that one.

You will learn the pattern, frequency and other limits of the GameBoy and the musicsystem. Together with the following possibilities. With Music Box player you can start the music at any pattern, stop at the end or loop to a given pattern position (what isn't always the beginning), this way as many subsongs can also be done as many the musician can squeeze into a Music Box file, the music also can be stopped manually any time (and continued too, but only with the Music Box 2 player) and soundeffects can be played any time on any given channel with any given instrument and on any given frequency (with halfnote accuracy) with arpeggio, vibrato and portamento.

Compilers

You also need a good programming knowledge and experience, especially about GameBoy. Even if beginners can use this system easily, it is really recommended. Also it is recommended to study the examples available in this package and to have RGBDS, ISAS or GBDK compiler with a good knowledge about it. GBDK is working with RGBDS perfectly and other compilers aren't really recommended, so in this documentation we mostly refer to the RGBDS and ISAS solutions (these are the best and most common GameBoy compilers). Anyway after understanding this booklet and the examples, everybody can use the player in his/her favourite compiler too if it is different from these and if he/she knows it well. There are only minor differences between different GameBoy compiler formats, brackets, label styles, org/section usage, formats and limitations of hexadecimal, decimal and binary bytes, access of the hardware registers and the format and possibility to include binary files or anything. (Only the worst compilers are really strange with too many limitations.) So if you are programming in your own compiler, you sure can convert the few lines into it, if you aren't, let us recommend you the freely available RGBDS, can be downloaded from <http://hjem.get2net.dk/surfsmurf/>.

Including The Player and Music

The Music Box 1 player available in this package is a precompiler binary program. There are many reasons for this, the main reason is that this way the player can be used with absolutely every GameBoy compiler, including the ones will be done in the future. (The package before version 1.03 only had an RGBDS library, what wasn't useful with other compilers.)

The Music Box 2 package customers get when they purchase the system of course including the possibility for the developers to compile the player to a similiar binary file everywhere into the memory (ROM, SRAM, WRAM) and to set the player's memory usage everywhere (SRAM, WRAM).

Anyway, the precompiled PLAYER.BIN can be found in the freely spreaded package is also really useful. It was compiled into the WRAM, into $d000, also using the memory immediately after the player in the same area. It is extremely useful for the hobby programmers, it doesn't take memory from the home ROM bank as the player was available in the packages before version 1.03. It is only using WRAM and that area where the WRAM has several banks in the GameBoy Color (especially useful with GameBoy Color), the WRAM bank with the player have to be enabled only when the player functions are called and the other times the whole WRAM area can be used for anything else. When the player is precompiled into a ROM area, it has to be included to the original area where it is starting (for example if it was compiled into $3000, it have to be included there with org or section), when the player is precompiled into RAM as in this case, it can be included everywhere, but before using it, the program has to copy it manually into its final place into the SRAM/WRAM.

Both RGBDS and ISAS can include binary files, therefore can include the player too, easily. Some older compilers can only include sources. In these cases the binary files (player, music, etc) have to be recompiled into a byte list what can be included into the assembler source (and some assemblers can’t include, so cut and paste is needed into one source file) and can be compiled later back into binary. In these cases you have to watch out, some compilers are accepting only decimal byte lists or strange hexadecimal prefixes and postfixes. There are millions and millions of these bin2asm and bin2c tools available freely, every programmer has some and can be programmed in a couple of minutes too, so probably it will not be a problem to use the player with such a limited compilers. And if it is, we are recommending the free RGBDS.

Incbin with RGBDS

With RGBDS the binary include is really easy. The following example is including the binary player and the binary music file (XAYNE.SAV), directly taken from a card’s SRAM, demonstrating how easy this process is. Just the incbin command followed by the binary filename in quotes. In this example both files get a label too, so later we can refer to these.

```
player:         incbin "player.bin"     ; the player as binary data


music:          incbin "xayne.sav"      ; the music as binary data
                                        ; this is a simple 8K file directly
                                        ; from the editor card's SRAM or
                                        ; saved by an emulator as
                                        ; musicbox.sav
```

Libbin with ISAS

The same isn’t more compicated with ISAS either. The only difference is the libbin command and it is simply followed by the binary filename without any extras.

```
player:         libbin player.bin       ; the player as binary data


music:          libbin xayne.sav        ; the music as binary data
                                        ; this is a simple 8K file directly
                                        ; from the editor card's SRAM or
                                        ; saved by an emulator as
                                        ; musicbox.sav
```

Memcopy with RGBDS

As it was stated above, to use the precompiled player available in this package, running at $d000 in WRAM, first you have to move it into its final area before using it. This is an easy process isn’t needed in versions compiled into ROM with Music Box 2 available for customers. Anyway for beginners here is the RGBDS solution to move the binary included player from anywhere to $d000 into the WRAM. (If you want to use a different WRAM bank, set the bank before this.) This routine is referring to the **player** label mentioned in the previous section, the destination address ($d000) and the size of the player to move ($0421).

```
                ld      hl,player       ; this routine will upload the $0421
                ld      de,$d000        ; large musicplayer into the WRAM
                ld      bc,$0421        ; from $d000
uploadloop:     ld      a,[hl+]         ; note: you can use several WRAM
                ld      [de],a          ; banks on GBC in this area, so
                inc     de              ; theorically this will not waste
                dec     bc              ; memory, but always switch on the
                ld      a,b             ; player's WRAM bank before calling
                or      c               ; player functions
                jr      nz,uploadloop
```

Memcopy with ISAS

And the same for ISAS. The only differences are the bracket styles.

```
                ld      hl,player       ; this routine will upload the $0421
                ld      de,$d000        ; large musicplayer into the WRAM
                ld      bc,$0421        ; from $d000
uploadloop:     ld      a,(hl+)         ; note: you can use several WRAM
                ld      (de),a          ; banks on GBC in this area, so
                inc     de              ; theorically this will not waste
                dec     bc              ; memory, but always switch on the
                ld      a,b             ; player's WRAM bank before calling
                or      c               ; player functions
                jr      nz,uploadloop
```

Initialization (player + $0000)

When the music and the player are in their right places in ROM/SRAM/WRAM, the player is in its right place where it was compiled to and the music anywhere, the player have to be initialized first, of course as every player. Don’t call any other player functions before doing this and do this for every new music file!

This is a really easy method. The Initialization function have to be called with three parameters. The Initialization function is always the player + $0000 offset, in the case of the original example, $d000. (And before calling this or any other player function, the CPU needs an access to both the music and player, so if any of these are on different ROM/SRAM/WRAM banks, the player and music banks have to be switched on.)

The following parameters are needed in the following registers, before calling the player’s Initialization function at $d000:

HL = address of the music

A = beginning pattern position

B = pattern position to loop to after the end ($ff) command

HL is the music address, in our example simply the **music** label. A is the beginning pattern position, what is usually $00, but this way many short musics can be stored into one music file and the second one can start for example at $10 and for that $10 can be stored here. B is the pattern position to jump to when the pattern order sequencer is reaching an $ff command in the first column, what is the loop/restart command ($fe can be also used, what is the stop command, in this case B register is ignored), this is usually the same as the A register, usually the music is looping back to its start, but composers can loop back to the main part of a music too, not to play a long intro part, etc.

Also note that this function is destroying the A, B and HL registers.

Initialization with RGBDS

As the whole process was mentioned in the previous lines, the initialization is really easy. In this case both A and B are $00, this can be changed. The **music** label from the previous examples is stored in HL as that is the label of the music, $d000 is the place where the player is.

```
                ld      hl,music          ; load music address to hl
                xor     a                 ; load beginning pattern pos to a (0)
                ld      b,a               ; load restart pattern pos to b (0)
                call    $d000             ; initialize the music
```

Initialization with ISAS

This time it is absolutely the same as the RGBDS version, anyway here it is.

```
                ld      hl,music          ; load music address to hl
                xor     a                 ; load beginning pattern pos to a (0)
                ld      b,a               ; load restart pattern pos to b (0)
                call    $d000             ; initialize the music
```

Stop Function (player + $0003)

This function will stop the player, the music and the audio output immediately when it is called. In the player available in the Music Box 1 package there is no Continue function, what can continue the music after this. In the case of this player version, after the Stop function only the Initialization function works from the player functions. So that is needed to restart the music, maybe a different submusic or at a different position.

This is a really easy to use function too. The Stop function have to be called without any parameters. The Stop function is always the player + $0003 offset, in the case of the original example, $d003. (And before calling this or any other player function, the CPU needs an access to both the music and player, so if any of these are on different ROM/SRAM/WRAM banks, the player and music banks have to be switched on.)

Also note that this function is destroying the A register.

Stop with RGBDS

As it was stated above, it is a really simple function, does only one thing, stopping the player, the music, the soundeffects and the whole audio output. Only a function call, nothing more.

```
                call    $d003             ; stop the music
```

Stop with ISAS

Of course it is totally the same with ISAS too, see below.

```
                call    $d003             ; stop the music
```

Play Function (player + $0006)

This function is the soul of the system. Have to be called most often and does the real playing. After the initialization process, this function have to be called periodically to update the music and soundeffects. The period also depends on the player type (single/double/quattro = 1X/2X/4X). As most of the similiar systems, this is also tied to the screen refresh and the usual musics are the singleplayer ones (using less CPU time per frame). Singleplayer means the Play function have to be called only once at each screen refresh. It absolutely isn’t important where the raster is when this is called, but that is important to be at the same position in every frame, otherwise the timing wouldn’t be exact (sometimes faster, sometimes slower). A good method is to tie a routine to the VBLANK interrupt what is calling the Play function every time the VBLANK interrupt is generated and the music is playing. (This state can be stored in a byte for the reason not to call the Play function when a music isn’t initialized.) As the VBLANK is an important area, it’s better to use all the VBLANK time on VRAM and OAM transfers, another good possibility is to use the player with the LYC interrupt. (What can be out from the VBLANK too, the only important thing is to call the music at the same raster position what can be reached easily with this.) Of course there is a less elegant method which doesn’t use interrupts and wasting the battery, but it is the smallest and easiest to understand by the beginners, that’s why this method is used in all the Music Box 1-2 examples, also in this documentation. The routine is checking the LY register ($ff44) and when it is indicating that the raster is in the given line, it is calling the Play function.

This is a really easy to use function too, even if its calling time is more critical and does a lot of things, only a call command without any parameters, nothing more. The Play function is always the player + $0006 offset, in the case of the original example, $d006. (And before calling this or any other player function, the CPU needs an access to both the music and player, so if any of these are on different ROM/SRAM/WRAM banks, the player and music banks have to be switched on.)

Also note that this function is destroying the AF, BC, DE and HL registers.

Play with RGBDS

This is a possible (quick and dirty, battery eater) mainloop to play and update single speed music and soundeffects. The most important part is the **call $d006** part, the others are just timing.

```
mainloop:       ld      a,[$ff44]       ; wait for a raster position, in this
                cp      $00             ; case for the 000th position, but
                jr      nz,mainloop     ; can be changed

                call    $d006           ; update the music in every frame at
                                        ; the same raster position

                jr      mainloop        ; jump back to the main loop
```

Play with ISAS

And the same with ISAS. The only differences are the bracket styles.

```
mainloop:       ld      a,($ff44)       ; wait for a raster position, in this
                cp      $00             ; case for the 000th position, but
                jr      nz,mainloop     ; can be changed
                call    $d006           ; update the music in every frame at
                                        ; the same raster position
                jr      mainloop        ; jump back to the main loop
```

Double and Quattro Player

This is another method for playing music, still using the same Play function. Double and Quattro playing is useful to reach smoother and faster music, a common method on Commodore 64. Also it is possible to make better timing in the music. It is nothing else than calling the music more than once per screen refresh. Therefore it needs twice or four times as much CPU time as the normal Single player calling method and using the CPU at many different areas on the screen. These methods have too many disadvantages with these, not too many rastertricks and others can be used and a really lot of CPU time is being wasted for the music, that’s why these methods are really not recommended for games and demos, only mostly for standalone music presentations or titlescreens and similiar areas when needed. When a music is composed with the 1X setting in the Extra Menu of the Music Box tracker (and using .1X extension usually), that is singleplayer, the 2X setting (and .2X extension) means double player and the 4X setting (and .4X extension) means quattro player.

Double player works the same as the single (same initialization, etc), but have to be called exactly at every half screen, quattro player have to be called exactly at every quarter screen. (Both including VBLANK.) Always at the same raster position. Therefore VBLANK interrupt cannot be used with these, only LYC with changing LYC destinations. Also the LY register checking loop works here too, so we demonstrate this method. Still it isn’t recommended as not an elegant solution and wasting the battery, but it is the fastest, smallest and most easy to understand method for the beginners.

And again, double and quattro players aren’t really recommended in games and demos, only where CPU time and timing isn’t an issue. Single speed is usually enough, so that is better.

Double Player with RGBDS

This is a possible (quick and dirty, battery eater) mainloop to play and update double speed music and soundeffects.

```
mainloop:       ld      a,[$ff44]       ; wait for a raster position, in this
                cp      $00             ; case for the 000th position, but
                jr      nz,mainloop     ; can be changed
                call    $d006           ; update the music
subloop:        ld      a,[$ff44]       ; wait for a raster position, in this
                cp      $4d             ; case for the 077th position, but
                jr      nz,subloop      ; can be changed
                call    $d006           ; update the music
                jr      mainloop        ; jump back to the main loop
```

Double Player with ISAS

And the same with ISAS. The only differences are the bracket styles.

```
mainloop:       ld      a,($ff44)       ; wait for a raster position, in this
                cp      $00             ; case for the 000th position, but
                jr      nz,mainloop     ; can be changed
                call    $d006           ; update the music
subloop:        ld      a,($ff44)       ; wait for a raster position, in this
                cp      $4d             ; case for the 077th position, but
                jr      nz,subloop      ; can be changed
                call    $d006           ; update the music
                jr      mainloop        ; jump back to the main loop
```

Quattro Player with RGBDS

This is a possible (quick and dirty, battery eater) mainloop to play and update quattro speed music and soundeffects.

```
mainloop:       ld      a,[$ff44]       ; wait for a raster position, in this
                cp      $00             ; case for the 000th position, but
                jr      nz,mainloop     ; can be changed
                call    $d006           ; update the music
subloop1:       ld      a,[$ff44]       ; wait for a raster position, in this
                cp      $27             ; case for the 039th position, but
                jr      nz,subloop1     ; can be changed
                call    $d006           ; update the music
subloop2:       ld      a,[$ff44]       ; wait for a raster position, in this
                cp      $4d             ; case for the 077th position, but
                jr      nz,subloop2     ; can be changed
                call    $d006           ; update the music
subloop3:       ld      a,[$ff44]       ; wait for a raster position, in this
                cp      $73             ; case for the 115th position, but
                jr      nz,subloop3     ; can be changed
                call    $d006           ; update the music
                jr      mainloop        ; jump back to the main loop
```

Quattro Player with ISAS

And the same with ISAS. The only differences are the bracket styles.

```
mainloop:       ld      a,($ff44)       ; wait for a raster position, in this
                cp      $00             ; case for the 000th position, but
                jr      nz,mainloop     ; can be changed
                call    $d006           ; update the music
subloop1:       ld      a,($ff44)       ; wait for a raster position, in this
                cp      $27             ; case for the 039th position, but
                jr      nz,subloop1     ; can be changed
                call    $d006           ; update the music
subloop2:       ld      a,($ff44)       ; wait for a raster position, in this
                cp      $4d             ; case for the 077th position, but
                jr      nz,subloop2     ; can be changed
                call    $d006           ; update the music
subloop3:       ld      a,($ff44)       ; wait for a raster position, in this
                cp      $73             ; case for the 115th position, but
                jr      nz,subloop3     ; can be changed
                call    $d006           ; update the music
                jr      mainloop        ; jump back to the main loop
```

Soundeffect Player Function (player + $0009)

Nor the Soundeffect Player function isn’t really complicated, even if it has the most parameters and a few things to keep in mind. This function can be called any time after the initialization, while the music is playing with the Play function. When it is called with the right parameters, the soundeffect will start to play. Soundeffects are the same as the instruments and are also using the arpeggio, vibrato and portamento effects. It is recommended to play the soundeffects on an empty channel what the music doesn’t use (otherwise soundeffects could interrupt some lead parts and also some notes in the music could shut down longer looping soundeffects earlier than it is needed). It is also recommended to have a null instrument to shut down the looping soundeffects. Also note that if you aren’t using any music and only soundeffects only on the fourth channel (what doesn’t have frequency, therefore no arpeggio, vibrato and portamento effects and notes), the Play function isn’t needed, in this (and only in this) case the Soundeffect Player function can be used without the Play function.

The Soundeffect Player function have to be called with three parameters. The Soundeffect Player function is always the player + $0009 offset, in the case of the original example, $d009. (And before calling this or any other player function, the CPU needs an access to both the music and player, so if any of these are on different ROM/SRAM/WRAM banks, the player and music banks have to be switched on.)

The following parameters are needed in the following registers, before calling the player’s

Soundeffect Player function at $d009:

A = instrument number ($00 is the first, what is $01 in the tracker)

B = frequency (with half note accuracy: $00 = C-3, $01 = C#3)

C = channel number ($00 = channel 1, $01 = channel 2)

The above lines are speaking for themselves. It is really important to keep in mind that $00 is the first instrument, what is numbered as $01 in the Music Box tracker and this way the instruments from the tracker have to be subtracted in A register by one. The frequency is halfnote based starting from C-3, what is $00, and increasing by one at every halfnote, these can be easily looked up in the Music Box tracker. And the channel number is ranging from $00 to $03, where $00 is the first channel, $01 is the second, etc. Also keep in mind here too (not just in the tracker) to play back instruments designed for channel 3 only on channel 3, etc.

Also note that this function is destroying the AF, BC, DE and HL registers.

Soundeffects with RGBDS

The following example will play a D#5 ($1b) note (so will play on the frequency can be calculated from this) with the instrument number $09 (what is $0a in the tracker) on the second channel ($01). The Play function calling isn’t demonstrated here, but of course that must be called at every frame (or more often depending on the replay speed the music and soundeffects were designed to), also the Initialization function is needed first, as always.

```
        ld      a,$09           ; a = $09, instrument $0a

        ld      b,$1b           ; b = $1b, note d#5

        ld      c,$01           ; c = $01, channel 2

        call    $d009           ; play the soundeffect
```

Soundeffects with ISAS

Of course it is totally the same with ISAS too, see below.

```
        ld      a,$09           ; a = $09, instrument $0a

        ld      b,$1b           ; b = $1b, note d#5

        ld      c,$01           ; c = $01, channel 2

        call    $d009           ; play the soundeffect
```

How to Buy Music Box

Music Box 1 can be freely used in any non-commercial projects, including mp3 music, live performance and free GameBoy demos and games. (In these cases all we need is your free work, the credit for us and a donation if you can afford and you think we deserve for our work.)

It also can be licensed into commercial projects (games, demos and tools), now available only together with Music Box 2. (So the package is including two trackers for one price, one of them is running on the GameBoy, one of them on the PC, with many tools, converters, examples, etc. Also with full and fast customer support and help. We answer every questions, make more examples and help out with every problems on request. We even make minor modifications and customizations to the system if some customers need these.) For details, contact us.

Another licensing possibility into commercial projects (games, demos and tools) is, to license only the musicplayer for a lower price and hire us to make all the audio works in the project including musics and soundeffects. Also contact us for details.

Copyright

Music Box, Music Box 1 and Music Box 2 trackers, players, fileformats, documentations and additional related tools were done by, copyrighted by and property of Zsolt Minier and Ray Nemes. All other mentioned names, copyrights and trademarks referred in this document are still the property of their respective owners (including but not limited to Nintendo, GameBoy, GameBoy Color, Super GameBoy, Wide Boy, GameBoy El Blacklight, Fasttracker II, etc). Modifying, reselling, disassembling of Music Box, Music Box 1, Music Box 2 trackers, players, fileformats and additional related tools are forbidden without a special written agreement from the authors. Music Box 1 package can be spreaded for free in (and only in) its original, unodified form as it is available at our website: <http://blackbox.resource.cx/> in a zip file. Music Box 1 system can be freely used for noncommercial purposes in noncommercial projects until credit is given to the authors. Other licenses are available from the authors.

Thanks to Ádám Ábrahám, Ákos Balázs, David S. Berkompas, Mr EMP, Collin van Ginkel, Jukka-Pekka Luukkonen, Balázs Oszvald, Martijn Reuvers, Attila Szõke, Péter Tihanyi and all the forgotten people for their help.

Ray Nemes and Zsolt Minier, <blackbox@resource.cx>