DS4QB++

Instruction Manual

Instruction Manual

DS4QB++

Version 1.2
Released March 12, 2002

By Chris Adams (aka Lithium) @ TRINTS Online © 2001-02

Table of Contents

Chapter 1 Introduction 5

Legal Stuff 5
What is DS4QB++? 5
Minimum System Requirements 5
Version Updates 6

Chapter 2 Getting Started 8

How does DS4QB++ work? 8
    DS4QB++ Architecture 8
    Communication System 8
    What sound/music formats does DS4QB++ support? 9

Adding DS4QB++ to your project 10
    Adding DS4QB++ to your program 10
    Customizing DS4QB++ for your program 11

Starting up and shutting down DS4QB++ 12
    Init and Deinit 12
    Global Volumes 13

Sound Engine: Basics 13
    Looping Sounds 13
    Playing, Stopping, etc. 13

Music Engine: Basics 14
    Loading Music 14
    Playing, Stopping, etc. 15

Chapter 3 Advanced Topics 17

Sound Engine: Advanced Topics 17
    Frequency, Panning, Volume, etc. 17
    Sound Buffers 18
    2D Audio 19

Music Engine: Advanced Topics 20
    Pan separation and volume 20
    Fading and fade-switching 21
    Positions 24

Modifying and compiling DS4QB++ 26
    What do I need? 26
    Modifying and compiling the slave module 26

Chapter 4 Individual Function Explanations 27

Base Functions 27
  DS4QB.Close 27
  DS4QB.SetGlobalVolumes 27
  DS4QB.SetMusic 27
  DS4QB.SetSound 28
  DS4QB.SetMasterVolume 28
  DS4QB.GetOs% (Function) 28
  DS4QB.Init% (Function) 29

Sound Functions 29
  DS4QB.DeleteSound 29
  DS4QB.LoadSound 30
  DS4QB.PlaySound2D 30
  DS4QB.PlaySound2DEx 31
  DS4QB.PlaySound 31
  DS4QB.PlaySoundEx 32
  DS4QB.SetSoundAttr 33
  DS4QB.StopSound 34

  DS4QB.AddSound 34
  DS4QB.Add2DSound 35
  DS4QB.PlaySounds 35
  DS4QB.Play2DSounds 36

Music functions 36
  DS4QB.DeleteMusic 36
  DS4QB.LoadMusic 36
  DS4QB.MusicFadeIn 37
  DS4QB.MusicFadeOut 38
  DS4QB.MusicFadeSwitch 38
  DS4QB.PauseMusic 39
  DS4QB.PlayMusic 39
  DS4QB.ResumeMusic 40
  DS4QB.SetMusicAttr 40
  DS4QB.SetMusicPosition 40
  DS4QB.StopMusic 41
  DS4QB.GetMusicLength& (Function) 41
  DS4QB.GetMusicPosition& (Function) 42
  Combine& (Function) 42
  GetOrder& (Function) 42
  GetRow& (Function) 43

CD Audio Functions 43
  DS4QB.InitCD 43
  DS4QB.DeinitCD 44
  DS4QB.PlayCD 44
  DS4QB.GetCDTracks (Function) 44

    DS4QB.GetCDTrackLength& (Function) 45

Generic 2D Functions 45
    DS4QB.Set2DPosition 45
    DS4QB.Set2DDistFactor 45

Other Functions 46
    RawExtract 46

Closing Words 47

Contact Information 47
Credits 47
Future Plans 47

Appendices 48

Constants 48
    Init Flags 48
    Sound Flags 48
    Music Flags 48
    Constants 49
    DEFAULT values 49

CHAPTER 1

Introduction

Legal Stuff

Please read the terms and conditions below. If you do not agree with one or more of these terms then please do not use DS4QB++!

- Use at your own risk! I will not be held liable for any damages caused to anything, by ANY file contained within the main DS4QB++ package, or the utilities package. Now I highly doubt it will cause any harm to your system. But if it does, it's probably 'cause you were a lazy ass and forgot to read the docs before you started :P
- You may edit ANY of the sourcecode in the library to your liking. However, if you wish to redistribute it with the modifications you made, please leave DS4QB++ somewhere in the name... For example: DS4QB++ X Edition
- Please give credit where it is due!

Now that that's over with…

What is DS4QB++?

DS4QB++ is a sound engine for Microsoft QuickBasic 4.5/7.1, which brings the power of DirectSound/BASS.DLL to the hands of your QB program! DS4QB++ is fast, easy to use, and has lots of features. DS4QB++ is the third installment in the "DS4QB" line. The first (DS4QB) was made by nodtviedt and the second (DS4QB2) was made by Nethergoth.

Minimum System Requirements

- Microsoft Windows 95
- Microsoft DirectX 6.1 or higher
- 486DX4/100 MHz
- 16 MB of RAM
- 512 KB of free disk space
- 32-bit disk access /w caching enabled

Version Updates

February 3rd, 2002

- 14 routines added:

  DS4QB.PlaySound2D
  DS4QB.PlaySound2DEx
  DS4QB.AddSound
  DS4QB.Add2DSound
  DS4QB.PlaySounds
  DS4QB.Play2DSounds
  DS4QB.InitCD
  DS4QB.DeinitCD
  DS4QB.PlayCD
  DS4QB.Set2DPosition
  DS4QB.Set2DDistFactor
  DS4QB.GetCDTracks (Function)
  DS4QB.GetCDTrackLength& (Function)

  RawExtract (Other)

- 1 routine removed:

  CrashFix (Debugging tool)

- 2D Audio support (software)

- Added 2 new example programs:
  CDDEMO.BAS
  2DDEMO.BAS

- OGG Vorbis support for music purposes.

- Made LITE and FULL versions of this library.

- Sound buffers, to allow the playing of multiple sounds in one master to slave data transfer.

- Major bugs fixed; your QB program won't crash anymore if too many sounds are played at once.

- Made a nifty little utility manager for all the utilities included with DS4QB++.

- No more Soundsys.dat. It was causing too many problems on NT/XP machines; I've replaced it with a subdirectory.

- RAW file routines for Sound/Music/Maps/Graphics/whatever.

- The **CrashFix** subroutine has been flunked in favor of an executable... For those times when your program crashes but the DS4QB++ slave is still running.

- More? Most likely, I fixed/tweaked a lot of things.

August 11th, 2001

- 7 routines added:

  DS4QB.GetMusicPosition& (Function)
  DS4QB.GetMusicLength& (Function)
  DS4QB.SetMusicPosition
  Combine& (Function, for MOD type music)
  GetOrder& (Function, for MOD type music)
  GetRow& (Function, for MOD type music)

  CrashFix (Debugging tool)

- FULL Mp3 support for music purposes.

- Changes made to a sound through **DS4QB.SetSoundAttr** can now be heard WHILE the sound is playing.

- DEFAULT can be used **in DS4QB.SetSoundAttr** and **DS4QB.PlaySoundEx** to use the sound's original frequency.

- Fixed bug in **DS4QB.SetMasterVolume** and **DS4QB.SetGlobalVolumes—** they actually work now!

- New demo program "QB Mp3 Player".

- Updated MpTrack. (utilities)

CHAPTER 2

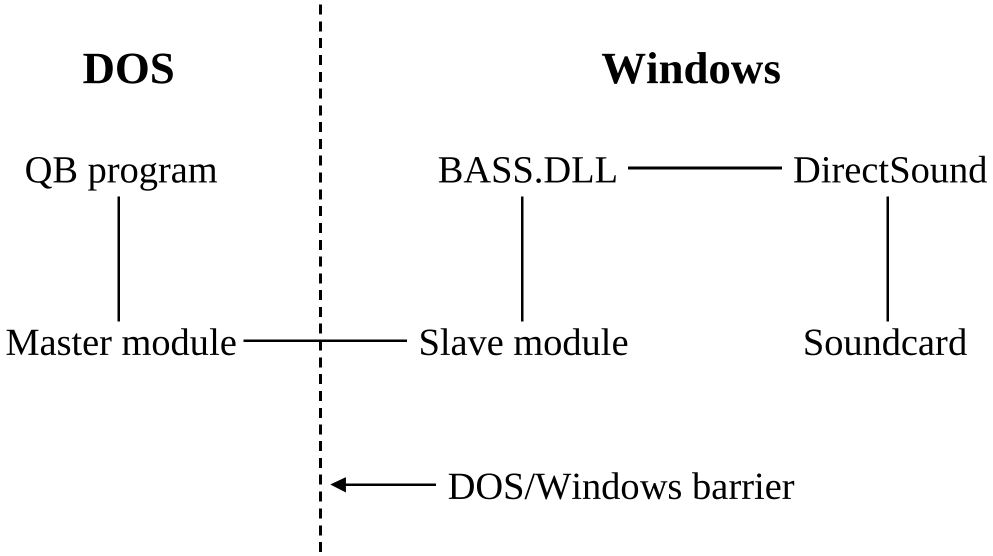
Getting Started

How does DS4QB++ work?

DS4QB++ Architecture

DS4QB++ is made up of two modules: the master module and the slave module. The DS4QB++ slave module is a Win32 application written in Visual C++ 6, and the master module is a DOS16 application written in QuickBasic 4.5.

The master module sends information to the slave module. The slave module then carries out its task and informs the master module it has done so. For instance, when you call DS4QB.LoadSound, the master module sends the loading information to the slave module, which in turn receives the information, loads the sound, and then tells the master module it is done. All this time the slave module is virtually "invisible", running in the background.



Communication System

DS4QB++ has two types of master to slave communication; the method used depends on the operating system running.

Windows 9x and Millennium (Me)

For Windows 9x/Me, DS4QB++ uses what is called "Swap files /w DMA 0 queuing". What does this mean? Simple: the master module creates a swap file and in it places all the information the slave module will need. It then sends a signal through DMA 0. Meanwhile, the slave module has been waiting for the signal to appear. When it does, the slave module reads the information from the swap file, carries out the task at hand, and then clears DMA 0 as a signal to the master module that it is ready for the next command.

This method is quite fast, and can transmit **large** amounts of data in no time at all.

Windows NT/2k/XP

Unfortunately the DMA 0 "trick" does not work under these operating systems (for a number of reasons). So DS4QB++ uses a slightly different method of communication called "Swap files /w file indicant queuing".

This works in much the same way as the Win9x/Me method, except it uses "file indicants" for queuing. What does this mean? After writing the packet data to the swap file, the master module announces the presence of the packet by creating a file. Meanwhile, the slave module has been waiting for this file to exist. After seeing the file, the slave module performs its task, and then deletes the indicant file as a sign to the master module that it has completed its task. This method is not quite as fast as the Win9X one, but it is still fast, and it gets the job done quite nicely.

What sound/music formats does DS4QB++ support?

For the sound engine:

- .WAV
- .MP1
- .MP2
- .MP3

Huh? Aren't .MP3’s for music?

No, .MP3’s are just compressed .WAV’s, although they have been heavily used for music purposes. You can compress all your .WAV files into .MP3 files using a converter I have included. .MP3s are just as good as .WAV’s for your sound effects. They won't take any more time to play, but will take longer to load (since they are decompressed at load).

For the music engine:

- .MOD
- .IT
- .XM
- .S3M
- .MTM
- .MO3
- .MP3
- .OGG

What, no .MID for the music engine?!

I have included a program that will convert your .MID files into .MOD, but I will probably not be adding direct .MID support to DS4QB++, as .MID's are a dying breed.

What is an .MO3 file?

An .MO3 file is a .MOD/.IT/.XM/.S3M/.MTM file with MP3 compressed samples, which can be up to 90% smaller. (Wow!) This is useful if you have .MOD files with really big samples, or maybe you converted a .MID file to a .MOD, and you want to make the file as small as it was before you converted it. .MO3 files don't play any slower than the other music formats; they just take longer to load.

What is an .OGG file?

OGG Vorbis, it's a new sound compression format that seems to have become quite popular. It offers higher quality and compression rates.

(All these converters can be found in the \UTILS directory.)

Adding DS4QB++ to your project

Adding DS4QB++ to your program

Step 1: Decide which version of DS4QB++ you're going to use:

LITE	Basic sound/music routines only
FULL	Everything

Step 2: Copy these files into your project directory:

LIB\\(*version)*\DS4QBPP.BAS	Master module
LIB\\(*version)*\DS4QBPP.BI	Master module internal header file
LIB\\(*version)*\DEXTERN.BI	Master module external header file

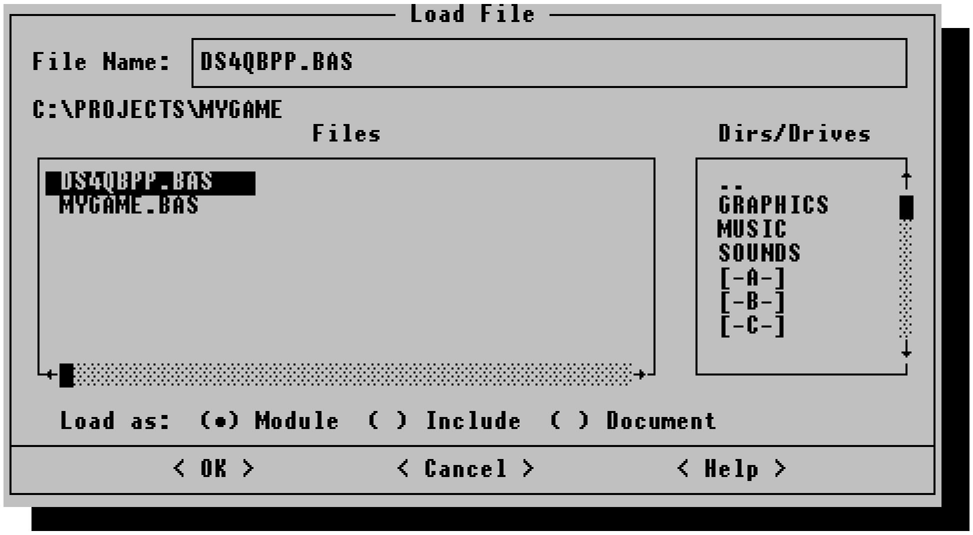
(*version* is the version you selected in step 1)

Step 3: Copy the SOUNDSYS directory into your project directory. Copy the directory itself, not just the files inside it. If you're not using Ogg Vorbis, then you can delete the files OGG.DLL and VORBIS.DLL from your SOUNDSYS subdirectory.

Step 4: Add this line near the start of your main source file:

```
'$INCLUDE: 'DEXTERN.BI'
```

Step 5: Load DS4QBPP.BAS as a module in your project:



File Name: DS4QBPP.BAS

C:\PROJECTS\MYGAME

Load as: (•) Module

Customizing DS4QB++ for your program

DS4QB++ comes with a SETUP program for use with your own program. It is a text mode GUI that allows the user to change the sound options for your program with ease. However, the setup program does not come compiled—you must configure and compile it yourself.

To configure the setup program, open SETUP.BAS (which is in the \SETUP directory you unpacked DS4QB++ into) and find the line which reads:

```
CONST PROGRAMNAME = "Your program's name here"
```

Place your program's name between the ""s. So for MyGame you would enter:

```
CONST PROGRAMNAME = "MyGame"
```

After doing this, save your new custom SETUP.BAS into your project directory, and compile it. (You now have your very own custom setup program for your application). Of course, you can change anything you wish in this program to fit your needs. You could add a section for the user to select game controls, or network settings for a network game—or maybe just change the colors/look of the windows. But be warned: This is *very* crappy coding! (VERY).

Starting up and shutting down DS4QB++

Init and Deinit

The very first thing you must do before calling any DS4QB++ routines is initialize DS4QB++. This can be done by calling DS4QB.Init:

```
' Initialize DS4QB++ using the sound quality specified in
SOUNDSYS.CFG,
' and using the default initialization flags
Result = DS4QB.Init(CURRENT, DEFAULT)
```

or maybe:

```
' Initialize DS4QB++ with high sound quality, and use the default
' initialization flags
Result = DS4QB.Init(HIGHQUALITY, DEFAULT)
```

If **DS4QB.Init** returns a non-zero value, there was an error. The error codes are as follows:

Error code	Description
-1	A critical file is missing from the SOUNDSYS directory (**DS4QBXX.EXE**, **BASS.DLL**, **START.EXE**, or **VORBIS.DLL**/**OGG.DLL** if you used the INIT.OGGENABLE flag)
-2	Could not connect to slave. This could be a result of an error occurring while the slave was attempting to initialize your soundcard.
-3	Configuration file (**SOUNDSYS.CFG**) is missing. This could be because **SETUP.EXE** has not been executed successfully.

The very last thing you must do (usually before your program terminates) is deinitialize DS4QB++. This will tell the slave module to shutdown and delete any excess files created by DS4QB++ while the program was operating. This is done by one call to **DS4QB.Close**:

```
' Deinitialize DS4QB++
DS4QB.Close
```

Global Volumes

Although it is not necessary to set global volumes, it can be very helpful for many things. There are two commands that can be used to alter these global volumes:

DS4QB.SetMasterVolume	Sets the master volume, which is a percentage over all the other volumes. 50 (default) is 100%, 100 is 200%, 25 is 50%, etc.

```
' Sets the master volume to 200%
DS4QB.SetMasterVolume 100
```

DS4QB.SetGlobalVolumes	Sets the individual global volumes over the different devices (sound and music). 50 is the default for both.

```
' Set sound volume to 150%, and leave the music volume at what
' it is now.
DS4QB.SetGlobalVolumes 75, CURRENT
```

Sound Engine: Basics

Loading Sounds

DS4QB++ can have up to 1024 sounds loaded at a time, which can all be playing simultaneously a maximum of 10 times each. But before we play any sounds we must load them. This can be done with **DS4QB.LoadSound**:

```
' Load MySound.Wav in to sound slot 1, using the default sound flags
DS4QB.LoadSound 1, "MySound.Wav", DEFAULT
```

Note There are no error messages. If the sound was not loaded correctly you just won't hear any sound when you play it.

Playing, Stopping, etc.

Some of DS4QB++'s sound handling routines include:

DS4QB.PlaySound	Play a sound
DS4QB.StopSound	Stop a sound from playing
DS4QB.DeleteSound	Delete a sound from memory

Here is a small program that shows how to use all these routines:

```
' This program demonstrates the basic DS4QB++ sound routines
DEFINT A-Z

'$INCLUDE: 'DEXTERN.BI'

' Initialize DS4QB++
IF DS4QB.Init(CURRENT, DEFAULT)) THEN
  PRINT "Error! Could not initialize DS4QB++!"
  END
END IF

' Load MyWav.Wav into slot 1
DS4QB.LoadSound 1, "MyWav.Wav", DEFAULT

DO
  Inpt$ = UCASE$(INPUT$(1))
  SELECT CASE Inpt$
    CASE "P"
      ' Play sound at slot 1
      DS4QB.PlaySound 1
    CASE "S"
      ' Stop sound at slot 1. If no sound is
      ' playing, nothing will happen
      DS4QB.StopSound 1
    CASE "D"
      ' Delete sound at slot 1, oops!
      DS4QB.DeleteSound 1
      PRINT "Oops!"
    CASE CHR$(27)
      EXIT DO
  END SELECT
LOOP

' Deinitialize DS4QB++
DS4QB.Close

END
```

Music Engine: Basics

Loading Music

DS4QB++ can have a maximum of 512 musical pieces loaded at any given time, which can all be playing at the same time. At the moment, DS4QB++ supports .MOD .MO3 .IT .XM .S3M .MTM .OGG and .MP3 music types. Loading a musical piece can be done with **DS4QB.LoadMusic**, like this:

```
' Load MyMusic.Mod into music slot 1, with the default music flags.
DS4QB.LoadMusic 1, "MyMusic.Mod", DEFAULT
```

Playing, Stopping, etc.

The basic music handling routines are as follows:

DS4QB.PlayMusic	Play a musical piece
DS4QB.StopMusic	Stop a musical piece from playing
DS4QB.PauseMusic	Pause a musical piece in mid-play
DS4QB.ResumeMusic	Resume a musical piece from pause status
DS4QB.DeleteMusic	Delete a musical piece from the memory

Here is an example demonstrating DS4QB++'s basic music routines:

```
' This program demonstrates the basic DS4QB++ music routines
DEFINT A-Z

'$INCLUDE: 'DEXTERN.BI'

' Initialize DS4QB++
IF DS4QB.Init(CURRENT, DEFAULT)) THEN
  PRINT "Error! Could not initialize DS4QB++!"
  END
END IF

' Load MyMusic.Mod into slot 1
DS4QB.LoadMusic 1, "MyMusic.Mod", DEFAULT

' Play music at slot 1, NOTE: By default musical pieces are
' automatically looped (When the music finishes playing it starts
' over and plays from the start)
DS4QB.PlayMusic 1

DO
  Inpt$ = UCASE$(INPUT$(1))
  SELECT CASE Inpt$
    CASE "P"
      ' Play music at slot 1
      DS4QB.PlayMusic 1
    CASE "A"
      ' Pause music at slot 1. If no music
      ' is playing, nothing will happen
      DS4QB.PauseMusic 1
    CASE "R"
      ' Resume a paused musical piece at slot 1, nothing will happen
      ' if no music has been paused
      DS4QB.ResumeMusic 1
    CASE "S"
      ' Stop music at slot 1. If no music
      ' is playing, nothing will happen
      DS4QB.StopMusic 1
    CASE "D"
      ' Delete music at slot 1, oops!
      DS4QB.DeleteMusic 1
      PRINT "Oops!"
    CASE CHR$(27)
      EXIT DO
  END SELECT
```

```
LOOP

' Deinitialize DS4QB++
DS4QB.Close

END
```

CHAPTER 3

Advanced Topics

Sound Engine: Advanced Topics

Frequency, Panning, Volume, etc.

What are they? First of all, the frequency is the rate at which the sound plays. If a sound was recorded at 22 KHz, then when it is played you will hear 22,050 vibrations per a second. The higher the frequency, the better quality the sound…right? Well, that depends. As long as the sound was initially recorded at that rate, it will sound fine. But if a sound was recorded at 22 KHz and then played at 44 KHz, it will play twice as fast as it should! (And vice versa.)

So what is changing the frequency good for? It can be handy for many of different things: one sound played at different frequencies could become 2 sounds, or more! (This could be useful for simulating a car engine.)

So how do I play a sound at a different frequency? Easy, using the **DS4QB.PlaySoundEx** routine:

```
' Play sound at slot one, on using a random frequency between 100Hz
' and 44,100Hz, using the current values for volume, panning, and
' looping
DS4QB.PlaySoundEx 1, INT(RND * 44000) + 100, CURRENT, CURRENT, _
CURRENT
```

Of course, **DS4QB.PlaySoundEx** can do more than play a sound at a different frequency. It can also play a sound with different panning settings, or at a different level of volume, or with a different looping flag.

Panning is quite simple. It is the position from left to right on your speakers. 0 is centered (and default), -100 is completely to the left, and 100 is completely to the right.

```
' Play sound at slot one, using the current frequency for that sound
' effect, a random panning setting from -100 to 100, and using the
' current settings for volume and looping
DS4QB.PlaySoundEx 1, CURRENT, INT(RND * 200) - 100, CURRENT, CURRENT
```

DS4QB.PlaySoundEx can also play your sounds with different volumes. 0 is off, 50 is default, and 100 is full volume.

```
' Play sound at slot one, using current frequency, panning, and
' looping settings, with random volume between 0 and 100
DS4QB.PlaySoundEx 1, CURRENT, CURRENT, INT(RND * 101), CURRENT
```

And last but not least, you can play your sound with a different looping setting. ACTIVE is looping, DEACTIVE (default) is non-looping. When a sound is played with the looping setting on, the sound will continue to loop until either the sound is stopped (using **DS4QB.StopSound**) or DS4QB++ is deinitialized.

```
' Play a sound using the current settings for frequency, panning,
' and volume. Activate looping
DS4QB.PlaySoundEx 1, CURRENT, CURRENT, CURRENT, ACTIVE
' Wait for user to press a key
WHILE INKEY$ <> "": WEND
```

DS4QB.PlaySoundEx is very useful, and is just as fast as **DS4QB.PlaySound**. But what if I want to change any of these CURRENT settings? You can change any of these CURRENT settings using **DS4QB.SetSoundAttr**:

```
' Set attributes for sound slot one, don't change the frequency, set
' volume to 100, panning to 50% to the left, don't change the
' looping flag, and change the sound flags so the sound plays at
' 8-bit/sample
DS4QB.SetSoundAttr 1, CURRENT, 100, -50, CURRENT, SND.8BIT
```

To set the attributes to default:

```
' Restore the default settings for volume, panning, looping, and
' flags
DS4QB.SetSoundAttr 1, DEFAULT, DEFAULT, DEFAULT, DEFAULT, DEFAULT
```

Note Setting DEFAULT for frequency will restore it to what it was when the sound was loaded.

Sound Buffers

Q. What is a sound buffer?
A. Very useful ;]

Sound buffers are VERY useful. DS4QB++ can play from 30 to 50 sounds/second; now let's say you want to be able to play more than 50 sounds/second. No problem, just use sound buffers! Basically, they allow you to play multiple sounds in one master to slave transaction.

For example, to add a sound to the sound buffer:

```
' Add sound 1 to the regular sound buffer, using CURRENT settings
' for the frequency, volume, panning, and looping.
DS4QB.AddSound 1, CURRENT, CURRENT, CURRENT, CURRENT
```

I could add another sound if I wanted:

```
' Add sound 3 to the regular sound buffer, using 10,000 for the
' frequency, 50 for the volume, and CURRENT settings for the panning
' and looping.
DS4QB.AddSound 3, 10000, 50, CURRENT, CURRENT
```

I could add another 62 sounds to this buffer... But I won't. These sounds will not be played until **DS4QB.PlaySounds** is called. After the sounds are played, the buffer will be cleared. There is also a sound buffer for 2D sounds, which is explained in the next section.

2D Audio

First off, what is 2D Audio? Well, simply put, the volume and pan position of a sound are altered to simulate distance, position, and orientation from the perception (ear) point, on a 2D plane. The 2D audio will work on any system that normal sound routines work on.

The only thing you really need to do before using the 2D audio routines is set the distance factor. Distance what? Basically, the 2D audio routines determine the volume of the sound using this formula:

$v = c / d^2$

or

Volume = OriginalVolume / (Distance * Distance)

Volume and OriginalVolume are pretty self explanatory, Distance also; it's the distance from the point of perception (the ear of the listener)... Anyway, most of you are probably using pixels as the unit of measure for distance, so if the original volume is at 100, and the distance is 10 pixels from the P.O.P:

$= 100 / 10^2$
$= 100 / 100$
$= 1$

1 volume is almost silent, that's not good :]. This is where the distance factor comes in handy, and the **DS4QB.Set2DDistanceFactor** routine. Let's say we set the distance factor to 64, and we're playing a sound from 10 pixels away, with the initial volume of 100:

$= 100 / (10^2 / 64)$
$= 100 / (100 / 64)$
$= 64$

Much better, no?

```
' Set the 2D distance factor to 64 (the DEFAULT setting)
DS4QB.Set2DDistanceFactor 64
```

There are two methods you can use to play sounds in 2D; the first is to play the sounds individually. The second is to use the 2D sound buffer, which is the preferred.

When playing sounds individually, you must first set the position and orientation of the ear, with **DS4QB.Set2DPosition**.

```
' Set the position of the ear to (10, 10), at an angle of 90 degrees
DS4QB.Set2DPosition 10, 10, 90
```

And **DS4QB.PlaySound2D** or **DS4QB.PlaySound2DEx** to play a sound:

```
' Play sound 1, at coordinates (0, 20), angle, using CURRENT values
' for the frequency and volume
DS4QB.PlaySound2DEx 1, 0, 20, 270, CURRENT, CURRENT
```

Although the option to play the sounds individually is given, it is not advised. It is a much better idea to use the 2D sound buffer, as you can play all your sounds for each frame at the same time and set the position/orientation of the ear all in one master to slave communication. A lot faster than doing it individually.

Simply add a sound to the sound buffer

```
' Add sound #1 to the 2D sound buffer, position (30, 50), angle 180,
' etc..
DS4QB.Add2DSound 1, 30, 50, 180, CURRENT, CURRENT
```

And maybe another (up to 60 2D sounds can be queued in the buffer)...

```
' Add another sound...
DS4QB.Add2DSound 2, 100, 20, 45, CURRENT, 50
```

Finally, play all the sounds at once using **DS4QB.Play2DSounds**

```
' Play all sounds in the 2D sound buffer, at position (10, 10),
' angle 90.
DS4QB.Play2DSounds 10, 10, 90
```

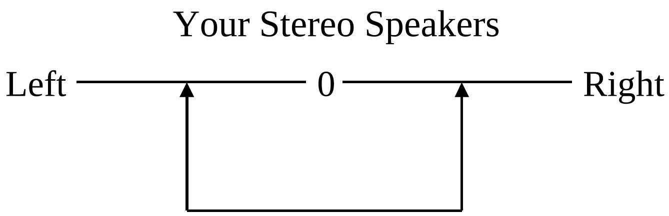
For a better example on this topic, check out DEMO\2DDEMO.BAS

Music Engine: Advanced Topics

Pan Separation and Volume

DS4QB++'s music engine has a variety of neat features, all of which can be executed with the same speed as DS4QB++'s sound routines!

First off is pan separation. This is how far your musical piece is spread out on your stereo speakers.



(Pan separation at 50%)
Music will be played between these points.

By default, the pan separation is set to 50%, but if you want a full spread, you can set it to 100% (sounds cool!) This can be done by calling **DS4QB.SetMusicAttr**:

```
' Set music attributes on slot one, with 100% pan separation, and
' current volume settings
DS4QB.SetMusicAttr 1, 100, CURRENT
```

Setting the volume is just as easy. By default the volume is 50, but you can set it using **DS4QB.SetMusicAttr**:

```
' Set music attributes on slot one, leave pan separation at what it
' is, and set volume to 25
DS4QB.SetMusicAttr 1, CURRENT, 25
```

Note You can call **DS4QB.SetMusicAttr** while the musical piece is playing. However, if you want to set a musical piece's attributes to something other than the default, it's a good idea to call **DS4QB.SetMusicAttr** before you call **DS4QB.PlayMusic**! For example:

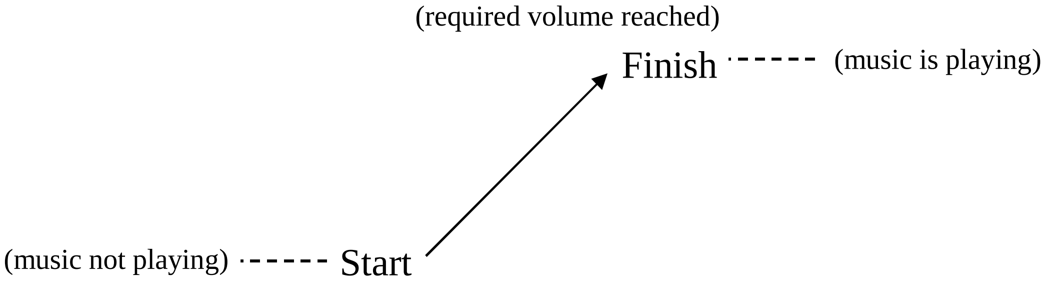
```
' This code is bad! The user will hear the music being played at the
' default volume for a split second, and then hear it go up
DS4QB.PlayMusic 1
DS4QB.SetMusicAttr 1, 100, CURRENT

' This is better
DS4QB.SetMusicAttr 1, 100, CURRENT
DS4QB.PlayMusic 1
```

Fading and Fade-Switching

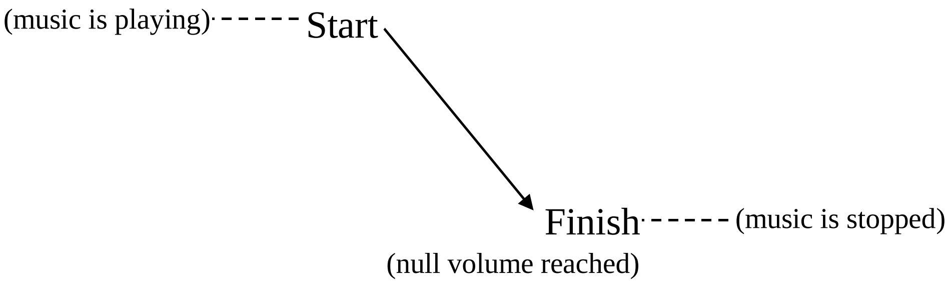
Fading and fade-switching are probably some of the coolest routines DS4QB++'s music engine has!

Fading is the process of gradually changing the volume from one level to another. DS4QB++ has three routines for this:

DS4QB.MusicFadeIn Starts a musical piece playing at null volume, and then gradually increases the volume until it has reached the required volume level.



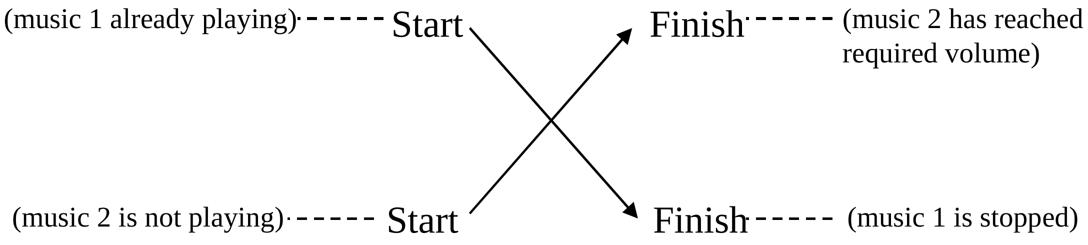
DS4QB.MusicFadeOut Fades a musical piece until it has reached null volume. Once it has, the music will stop playing.



Here is an example program that demonstrates the usage of both of these routines:

```
' This program demonstrates the usage of DS4QB.MusicFadeIn and
' DS4QB.MusicFadeOut
DEFINT A-Z

'$INCLUDE: 'DEXTERN.BI'

' Initialize DS4QB++
IF DS4QB.Init(CURRENT, DEFAULT)) THEN
  PRINT "Error! Could not initialize DS4QB++!"
  END
END IF

' Load MyMusic.Mod into slot 1
DS4QB.LoadMusic 1, "MyMusic.Mod", DEFAULT

' Start the fade in process for music slot 1, set the amount of
' volume to be increased/per a cycle to 5, and the volume that has
' to be achieved to 50.
' NOTE: FadeTemp is the variable that is used to track the progress
' of the fade. We call DS4QB.MusicFadeIn once in order to start the
' process, then we put it in the loop bellow to continue the process

DS4QB.MusicFadeIn 5, 1, 50, FadeTemp

' ALSO NOTE: The values 5 and 50 for the fade step and objective
' volume are default, I could have just written:
'
' DS4QB.MusicFadeIn DEFAULT, 1, DEFAULT, FadeTemp
'
' Sometimes it is best to use DEFAULT for attributes that can use
' it. (if you're not sure what to put, or you just want to keep
' things consistent)

DO
  ' Continue the process. I bet you're wondering, why do we need to
  ' do all this? It's simple; this way you can be doing other things
  ' while waiting for the music to fade in/out.
  DS4QB.MusicFadeIn DEFAULT, 1, DEFAULT, FadeTemp
  ' We know the fade process is over when FadeTemp is equal to zero
LOOP WHILE FadeTemp

' Prompt user and wait for a keypress
PRINT "Press any key to stop."
G$ = INPUT$(1)

' Initiate the fade out process for music slot 1, using the default
' settings for the fade-step and current volume. (The current volume
' is the volume that the musical piece you are fading out from is
' at, this is needed because retrieving the current volume from the
' slave module would take too much time.)
DS4QB.MusicFadeOut DEFAULT, 1, DEFAULT, FadeTemp

DO
  ' Continue the fade out process
  DS4QB.MusicFadeOut DEFAULT, 1, DEFAULT, FadeTemp
  ' Wait until the process is finished
LOOP WHILE FadeTemp

' Deinitialize DS4QB++
DS4QB.Close

END
```

Of course, DS4QB++ can do more than just fade music in and out. What if you wanted to switch music? DS4QB++ has a neat little routine to do that for you: **DS4QB.MusicFadeSwitch**. This routine works like **DS4QB.MusicFadeIn** and **DS4QB.MusicFadeOut**, except you have two musical pieces, the first one is one that is already playing, the second one is one that will be playing when the routine is finished switching.



The neat thing about this routine is that you can seamlessly switch musical pieces, and have other stuff going on while it is doing so! Here is an example program for **DS4QB.MusicFadeSwitch**:

```
' This program demonstrates the usage of DS4QB.MusicFadeSwitch
DEFINT A-Z

'$INCLUDE: 'DEXTERN.BI'

' Initialize DS4QB++
IF DS4QB.Init(CURRENT, DEFAULT)) THEN
  PRINT "Error! Could not initialize DS4QB++!"
  END
END IF

' Load MyMusic.Mod into slot 1
DS4QB.LoadMusic 1, "MyMusic.Mod", DEFAULT
' Load MyMusic2.Mod into slot 2
DS4QB.LoadMusic 1, "MyMusic2.Mod", DEFAULT

' Start music at slot 1 playing
DS4QB.PlayMusic 1

PRINT "Press any key to switch the music."
G$ = INPUT$(1)

' Start the fade switch process, using the default settings for
' fade-step and object volume. NOTE: In this routine the objective
' volume is the volume at which the music you are switching to will
' end up as, and the current volume of the music you are switching
' from
DS4QB.MusicFadeSwitch DEFAULT, 1, DEFAULT, 2, FadeTemp

DO
  ' Continue fade switching the musics, using the same options as
  ' before
  DS4QB.MusicFadeSwitch DEFAULT, 1, DEFAULT, 2, FadeTemp
  ' Wait for fade switch process to finish
LOOP WHILE FadeTemp

' Prompt user and wait for a keypress
PRINT "Press any key to stop."
G$ = INPUT$(1)

' Deinitialize DS4QB++
DS4QB.Close

END
```

If you need more examples on these routines, the DS4QB++ demo program is a great source. If you downloaded the demo, the source code can be found in the \DEMO\SOURCE directory.

Positions

With DS4QB++, you can get/set the position of a musical piece at any time. The same routines are used for both MOD and Mp3 type music, but the processes are different. MOD type music uses "orders" and "rows", Mp3 type music uses position in bytes.

DS4QB++ has 3 universal routines for positions (music):

DS4QB.SetMusicPosition	Sets position
DS4QB.GetMusicPosition& (Function)	Retrieves position
DS4QB.GetMusicLength& (Function)	Retrieves length

And 3 other routines specifically for MOD type music:

Combine& (Function)	Combines order/row into a LONG.
GetOrder& (Function)	Gets order from a LONG.
GetRow& (Function)	Gets row from a LONG.

To retrieve the length (in bytes) of an Mp3 type music:

```
' Retrieve the length in bytes, of the Mp3 music in slot 1
Length& = DS4QB.GetMusicLength&(1)
```

To retrieve the position (in bytes) of a playing Mp3 type music:

```
' Retrieve the position (in bytes) of the Mp3 music in slot 1
Position& = DS4QB.GetMusicPosition&(1)
```

Let's say you wanted to find out what percentage of the Mp3 has played. You can do it this way:

```
' Retrieve the percentage of the Mp3 music that has been played
Percent = (100 / DS4QB.GetMusicLength&(1)) *
DS4QB.GetMusicPosition&(1)
```

You can also set the position using DS4QB.SetMusicPosition, like this:

```
' Retrieve length
Length& = DS4QB.GetMusicLength&(1)

' Randomly set the music's position at slot one
DS4QB.SetMusicPosition 1, INT(RND * Length&)
```

Doing the above is a little more complicated with MOD type music, due to their formatting, but not really much harder.

```
' Retrieve the length of the MOD music at slot one
Length& = DS4QB.GetMusicLength&(1)

' Convert the length into order/row
Order = GetOrder&(Length&)
Row = GetRow&(Length&)
```

That will give you the last order and row that a note is played on; retrieving position is very similar:

```
' Retrieve the current position of the MOD music playing at slot one
Position& = DS4QB.GetMusicPosition&(1)

' Convert the position into order/row
```

```
Order = GetOrder&(Position&)
Row = GetRow&(Position&)
```

And setting the position:

```
' Set the position of the music playing at slot one to 4 : 10 (Order
: Row)
DS4QB.SetMusicPosition 1, Combine&(4, 10)
```

Note: With Mp3 type music, I have noticed that with some .MP3's you cannot set the position too far off from what it is currently playing at—or it will reset itself. This is a bug/issue with BASS.DLL, so please, complain to Ian Luck at http://www.un4seen.com :]

If you would like a better example on this topic, check out MP3PLAY.BAS, in the \DEMO\SOURCE directory.

Modifying and Compiling DS4QB++

What do I need?

For the master module you need Microsoft QuickBasic 4.5 or 7.1. For the slave module you need Microsoft Visual C++ 6.0 or better, the Microsoft DirectX 6.1 SDK or better, and finally BASS Library v1.3 available at http://www.un4seen.com.

Modifying and compiling the slave module

To open up the DS4QB++ slave module workspace, just double click DS4QBXX.DSW in the \SOURCE directory. After compiling, copy DS4QBXX.EXE into your \SOUNDSYS directory and you're all ready to go.

CHAPTER 4

Individual Function Explanations

Base Functions

DS4QB.Close

Prototype `DECLARE SUB DS4QB.Close ()`

Parameters None.

Definition Shuts down and closes out DS4QB++.

Comments Call this just before your program terminates.

DS4QB.SetGlobalVolumes

Prototype `DECLARE SUB DS4QB.SetGlobalVolumes (SoundVol AS INTEGER, MusicVol AS INTEGER)`

Parameters SoundVol – Global sound volume.

DEFAULT	Uses the default setting. (50)
CURRENT	Uses the current setting.
Integer 0 – 100	Volume level.

MusicVol – Global music volume.

DEFAULT	Uses the default setting. (50)
CURRENT	Uses the current setting.
Integer 0 – 100	Volume level.

Definition Sets the global volumes for sound and music.

Comments None.

DS4QB.SetMusic

Prototype `DECLARE SUB DS4QB.SetMusic (Switch AS INTEGER)`

Parameters Switch – Music on/off indicant.

ACTIVE	Turns music on.
DEACTIVE	Turns music off.

Definition Turns music on or off.

Comments If music is turned off, calls to **DS4QB.PlayMusic** will be ignored.

DS4QB.SetSound

Prototype `DECLARE SUB DS4QB.SetSound (Switch AS INTEGER)`

Parameters Switch – Sound on/off indicant.

ACTIVE	Turns sound on.
DEACTIVE	Turns sound off.

Definition Turns sound on or off.

Comments If sound is turned off, calls to **DS4QB.PlaySound** and **DS4QB.PlaySoundEx** will be ignored.

DS4QB.SetMasterVolume

Prototype `DECLARE SUB DS4QB.SetMasterVolume (Volume AS INTEGER)`

Parameters Volume – Master volume level.

DEFAULT	Default setting is used. (50)
Integer 0 – 100	Volume level.

Definition Sets the master volume (volume of everything).

Comments None.

DS4QB.GetOS% (Function)

Prototype `DECLARE FUNCTION DS4QB.GetOs% ()`

Parameters None.

Returns WIN9X – User has selected Windows 95, 98, or Me in SETUP.EXE.
WINNT – User has selected Windows NT, 2K, or XP in SETUP.EXE.

Definition Retrieves the operating system class selected by the user in SETUP.EXE.

Comments None.

DS4QB.Init% (Function)

Prototype `DECLARE FUNCTION DS4QB.Init% ()`

Parameters SoundQuality – The sound quality that your sound/music will play at.

DEFAULT	Use the default setting. (MEDIUMQUALITY)
CURRENT	Use the setting selected by the user in SETUP.EXE.
HIGHQUALITY	High playback quality.
MEDIUMQUALITY	Medium playback quality.
LOWQUALITY	Low playback quality.

Flags – Initialization options for DS4QB++ to use.

DEFAULT	Use the default settings.
INIT.8BIT	Use 8-bit mixing for all sound/music/etc.
INIT.MONO	Use mono mixing for all sound/music/etc.

(These settings can be combined using the OR operand.)

Returns Non-zero value indicates an error; look in the “Getting started: Starting up and shutting down DS4QB++” section for the list of error codes/meanings.

Definition Initializes DS4QB++.

Comments This routine must be called before any other DS4QB++ routines!

Sound Functions

DS4QB.DeleteSound

Prototype `DECLARE SUB DS4QB.DeleteSound (Slot AS INTEGER)`

Parameters Slot – Sound slot you wish to delete.

Integer 1 – 1024	Sound slot.

Definition Deletes a sound.

Comments This will do nothing if no sound is loaded in this slot.

DS4QB.LoadSound

Prototype `DECLARE SUB DS4QB.LoadSound (Slot AS INTEGER, FileName AS STRING, Flags AS LONG)`

Parameters Slot – Sound slot that you wish to load the sound into.

Integer 1 – 1024	Sound slot.

FileName – File name of the sound you wish to load.

String	FileName.

Flags – Options for the sound you are loading.

DEFAULT	Use the default sound options. (NULL)

(These settings can be combined using the OR operand.)

SND.8BIT	Sound will play using 8-bit instead of 16-bit.
SND.MONO	Sound will play using mono instead of stereo.
SND.LOOPING	Looping flag initially set.
SND.SOFTWARE	Use software mixing instead of hardware.

Definition Loads a sound, from a file into the desired slot.

Comments Loads WAV/MP1/MP2/MP3 files only. FileName can be a maximum of 64 characters in length.

DS4QB.PlaySound2D

Prototype `DECLARE SUB DS4QB.PlaySound (Slot AS INTEGER, X AS INTEGER, Y AS INTEGER, Angle AS INTEGER)`

Parameters Slot – Sound slot you wish to play.

Integer 1 – 1024	Sound slot.

X, Y – Location of sound to be played.

Integer ? – ?	Sound position.

Angle – Angle in which sound is facing.

Integer 0 – 359	Sound orientation.

Definition Plays a sound with 2D audio.

Comments Will do nothing if no sound has been loaded in this slot.

DS4QB.PlaySound2DEx

Prototype

```
DECLARE SUB DS4QB.PlaySound (Slot AS INTEGER, X AS INTEGER, Y AS
INTEGER, Angle AS INTEGER, Freq AS INTEGER, Volume AS INTEGER)
```

Parameters Slot – Sound slot you wish to play.

Integer 1 – 1024	Sound slot.

X, Y – Location of sound to be played.

Integer ? – ?	Sound position.

Angle – Angle in which sound is facing.

Integer 0 – 359	Sound orientation.

Freq – Frequency to play the sound at.

CURRENT	Use current setting.
DEFAULT	Use original setting.
Long integer 100 – 100,000	Frequency.

Volume – Volume to play the sound at.

CURRENT	Use current setting.
DEFAULT	Use the default setting. (50)
Integer 0 – 100	Volume level.

Definition Plays a sound with 2D audio, extra options.

Comments Will do nothing if no sound has been loaded in this slot.

DS4QB.PlaySound

Prototype

```
DECLARE SUB DS4QB.PlaySound (Slot AS INTEGER)
```

Parameters Slot – Sound slot you wish to play.

Integer 1 – 1024	Sound slot.

Definition Plays a sound.

Comments Will do nothing if no sound has been loaded in this slot.

DS4QB.PlaySoundEx

Prototype

```
DECLARE SUB DS4QB.PlaySoundEx (Slot AS INTEGER, Freq AS LONG, Volume
AS INTEGER, Pan AS INTEGER, Looping AS INTEGER)
```

Parameters Slot – Sound slot you wish to play.

Integer 1 – 1024	Sound slot.

Freq – Frequency to play the sound at.

CURRENT	Use current setting.
DEFAULT	Use original setting.
Long integer 100 – 100,000	Frequency.

Volume – Volume to play the sound at.

CURRENT	Use current setting.
DEFAULT	Use the default setting. (50)
Integer 0 – 100	Volume level.

Pan – Panning position to play the sound at.

CURRENT	Use current setting.
DEFAULT	Use the default setting. (0)
Integer -100 – 100	Panning position.

Looping – Looping flag.

CURRENT	Use current setting.
DEFAULT	Use the default setting. (DEACTIVE)
ACTIVE	Turn looping on.
DEACTIVE	Turn looping off.

Definition Plays a sound, with extra functionality.

Comments Will do nothing if no sound has been loaded in this slot.

DS4QB.SetSoundAttr

Prototype `DECLARE SUB DS4QB.SetSoundAttr (Slot AS INTEGER, Freq AS LONG, Volume AS INTEGER, Pan AS INTEGER, Looping AS INTEGER, Flags AS LONG)`

Parameters Slot – Sound slot you wish to set these attributes on.

Integer 1 – 1024	Sound slot.

Freq – Frequency to set as current

CURRENT	Use current setting.
DEFAULT	Use original setting.
Long integer 100 – 100,000	Frequency.

Volume – Volume level to set as current

CURRENT	Use current setting.
DEFAULT	Use the default setting. (50)
Integer 0 – 100	Volume level.

Pan – Panning position to set as current.

CURRENT	Use current setting.
DEFAULT	Use the default setting. (0)
Integer -100 – 100	Panning position.

Looping – Looping flag to set as current.

CURRENT	Use current setting.
DEFAULT	Use the default setting. (DEACTIVE)
ACTIVE	Turn looping on.
DEACTIVE	Turn looping off.

Flags – Flags to set as current.

CURRENT	Use current settings.
DEFAULT	Use default settings. (NULL)

(These options can be combined using the OR operand.)

SND.8BIT	Use 8-bit, instead of 16-bit.
SND.MONO	Use mono, instead of stereo.
SND.SOFTWARE	Use software for mixing, instead of hardware.

Definition Sets a sound's CURRENT attributes.

Comments None.

DS4QB.StopSound

Prototype `DECLARE SUB DS4QB.StopSound (Slot AS INTEGER)`

Parameters Slot – Sound slot you wish to stop from playing.

Integer 1 – 1024	Sound slot.

Definition Deletes a sound.

Comments Nothing will happen if no sound is playing on this slot.

DS4QB.AddSound

Prototype `DECLARE SUB DS4QB.AddSound (Slot AS INTEGER, Freq AS LONG, Volume AS INTEGER, Pan AS INTEGER, Looping AS INTEGER)`

Parameters Slot – Sound slot you wish to play.

Integer 1 – 1024	Sound slot.

Freq – Frequency to play the sound at.

CURRENT	Use current setting.
DEFAULT	Use original setting.
Long integer 100 – 100,000	Frequency.

Volume – Volume to play the sound at.

CURRENT	Use current setting.
DEFAULT	Use the default setting. (50)
Integer 0 – 100	Volume level.

Pan – Panning position to play the sound at.

CURRENT	Use current setting.
DEFAULT	Use the default setting. (0)
Integer -100 – 100	Panning position.

Looping – Looping flag.

CURRENT	Use current setting.
DEFAULT	Use the default setting. (DEACTIVE)
ACTIVE	Turn looping on.
DEACTIVE	Turn looping off.

Definition Adds a sound to the sound buffer.

Comments The regular sound buffer can hold no more than 64 sounds.

DS4QB.Add2DSound

Prototype

```
DECLARE SUB DS4QB.Add2DSound (Slot AS INTEGER, PosX AS INTEGER, PosY
AS INTEGER, Angle AS INTEGER, Freq AS LONG, Volume AS INTEGER)
```

Parameters Slot – Sound slot you wish to play.

Integer 1 – 1024	Sound slot.

PosX, PosY – Location of sound to be played.

Integer	Sound position.

Angle – Angle in which sound is facing.

Integer 0 – 359	Sound orientation.

Freq – Frequency to play the sound at.

CURRENT	Use current setting.
DEFAULT	Use original setting.
Long integer 100 – 100,000	Frequency.

Volume – Volume to play the sound at.

CURRENT	Use current setting.
DEFAULT	Use the default setting. (50)
Integer 0 – 100	Volume level.

Definition Adds a 2D sound to the 2D sound buffer.

Comments The 2D sound buffer can hold no more than 60 sounds.

DS4QB.PlaySounds

Prototype

```
DECLARE SUB DS4QB.PlaySounds ()
```

Parameters None

Definition Plays all sounds in the regular sound buffer, and then clears it.

Comments Will do nothing if no sounds are in buffer.

DS4QB.Play2DSounds

Prototype `DECLARE SUB DS4QB.Play2DSounds (PosX AS INTEGER, PosY AS INTEGER, Angle AS INTEGER)`

Parameters PosX, PosY – Location of the ear.

Integer ? – ?	Sound position.
CURRENT	CURRENT values.

Angle – Angle in which the ear is facing.

Integer 0 – 359	Sound orientation.
CURRENT	CURRENT value.

Definition Plays all sounds in the 2D sound buffer, and then clears it.

Comments Will do nothing if no sounds are in buffer.

Music Functions

DS4QB.DeleteMusic

Prototype `DECLARE SUB DS4QB.DeleteMusic (Slot AS INTEGER)`

Parameters Slot – Music slot you wish to delete.

Integer 1 – 512	Music slot.

Definition Deletes a music slot.

Comments This will do nothing if no sound is loaded in this slot.

DS4QB.LoadMusic

Prototype `DECLARE SUB DS4QB.LoadMusic (Slot AS INTEGER, FileName AS STRING, Flags AS LONG)`

Parameters Slot – Music slot that you wish to load the music into.

Integer 1 – 512	Music slot.

FileName – File name of the music you wish to load.

String	FileName.

Flags – Options for the music you are loading.

DEFAULT	Use the default music options. (MUS.LOOPING)

(These settings can be combined using the OR operand).

MUS.8BIT	Music will play using 8-bit instead of 16-bit. *
MUS.MONO	Music will play using mono instead of stereo. *
MUS.SOFTWARE	Use software mixing instead of hardware. *
MUS.LOOPING	Looping flag initially set.

(* MOD only)

Definition Loads music, from a file into the desired slot.

Comments Loads .MOD/.IT/.MO3/.XM/.MTM/.S3M/.MP3/.OGG files only. FileName can be a maximum of 64 characters in length.

DS4QB.MusicFadeIn

Prototype

```
DECLARE SUB DS4QB.MusicFadeIn (FStep AS INTEGER, Slot AS INTEGER,
ObjVol AS INTEGER, CPos AS INTEGER)
```

Parameters FStep – Amount of volume increased per a timed cycle (300ms).

DEFAULT	Use the default setting. (5)
Integer 1 – 50	(must be a multiple of the objective volume).

Slot – Music slot to be faded in.

Integer 1 – 512	Music slot.

ObjVol – Object volume, fade in will be complete when this volume has been reached.

DEFAULT	Use the default setting. (50)
Integer 1 – 100	Volume to be achieved.

CPos – Pointer to variable for keeping track of the fade-in process.

Integer variable. (reference to)

Definition Starts playing, and fades in a musical piece.

Comments For a complete tutorial on this routine, see Advanced Topics → Music:Advanced Topics → Fading and fade-switching.

DS4QB.MusicFadeOut

Prototype

```
DECLARE SUB DS4QB.MusicFadeOut (FStep AS INTEGER, Slot AS INTEGER,
ObjVol AS INTEGER, CPos AS INTEGER)
```

Parameters FStep – Amount of volume decreased per a timed cycle (300ms).

DEFAULT	Use the default setting. (5)
Integer 1 – 50	(must be a multiple of the starting volume).

Slot – Music slot to be faded out.

Integer 1 – 512	Music slot.

ObjVol – Starting volume of music you are fading out.

DEFAULT	Use the default setting. (50)
Integer 1 – 100	Starting volume.

CPos – Pointer to variable for keeping track of the fade-out process.
Integer variable. (reference to)

Definition Fades out a musical piece, and eventually stops it.

Comments For a complete tutorial on this routine, see Advanced Topics → Music: Advanced Topics → Fading and fade-switching.

DS4QB.MusicFadeSwitch

Prototype

```
DECLARE SUB DS4QB.MusicFadeSwitch (FStep AS INTEGER, StartSlot AS
INTEGER, ObjVol AS INTEGER, EndSlot AS INTEGER, CPos AS INTEGER)
```

Parameters FStep – Amount of volume decreased/increased per a timed cycle (300ms).

DEFAULT	Use the default setting. (5)
Integer 1 – 50	(must be a multiple of the start/objective volume).

StartSlot – Music slot to be faded out.

Integer 1 – 512	Music slot.

ObjVol – Volume the music that is being faded out starts at, and volume that the music that is being faded in will finish at.

DEFAULT	Use the default setting. (50)
Integer 1 – 100	Volume level.

EndSlot – Music slot to be faded in.

Integer 1 – 512	Music slot.

CPos – Pointer to variable for keeping track of the fade-out process.

Integer variable.	(reference to)

Definition Fades out a musical piece (eventually stopping it), playing and fading in another one.

Comments For a complete tutorial on this routine, see Advanced Topics → Music: Advanced Topics → Fading and fade-switching.

DS4QB.PauseMusic

Prototype `DECLARE SUB DS4QB.PauseMusic (Slot AS INTEGER)`

Parameters Slot – Music slot you wish to pause.

Integer 1 – 512	Music slot.
CDMUSIC	CD Audio handle.

Definition Pauses a music slot.

Comments This will do nothing if no sound is playing in this slot.

DS4QB.PlayMusic

Prototype `DECLARE SUB DS4QB.PlayMusic (Slot AS INTEGER)`

Parameters Slot – Music slot you wish to play.

Integer 1 – 512	Music slot.

Definition Plays a music slot.

Comments Will do nothing if no music is loaded in this slot.

DS4QB.ResumeMusic

Prototype `DECLARE SUB DS4QB.ResumeMusic (Slot AS INTEGER)`

Parameters Slot – Music slot you wish to resume from paused status.

Integer 1 – 512	Music slot.
CDMUSIC	CD Audio handle.

Definition Resumes a paused music slot.

Comments Will do nothing if no music is paused in this slot.

DS4QB.SetMusicAttr

Prototype

```
DECLARE SUB DS4QB.SetMusicAttr (Slot AS INTEGER, Volume AS INTEGER,
Pan AS INTEGER)
```

Parameters Slot – Music slot you wish to set these attributes on.

Integer 1 – 512	Music slot.
CDMUSIC	CD Audio handle.

Volume – Volume level to set as current.

CURRENT	Use current setting.
DEFAULT	Use the default setting. (50)
Integer 0 – 100	Volume level.

Pan – Pan separation to set as current.

CURRENT	Use current setting.
DEFAULT	Use the default setting. (50)
Integer 1 – 100	Pan separation.

Definition Sets a music slot's CURRENT attributes.

Comments Will do nothing if no music has been loaded.

DS4QB.SetMusicPosition

Prototype

```
DECLARE SUB DS4QB.SetMusicPosition (Slot AS INTEGER, Position AS
LONG)
```

Parameters Slot – Music slot you wish to set the position.

Integer 1 – 512	Music slot.
CDMUSIC	CD Audio handle.

Position – Position you wish the music slot to jump to/play from.

Combine&(Order, Row)	Use for MOD type music.
Long Integer 0-?	Use for Mp3 type music.

Definition Sets a music slot's play position.

Comments For a more in-depth tutorial, see: Advanced topics → Music: Advanced topics → Positions.

DS4QB.StopMusic

Prototype `DECLARE SUB DS4QB.StopMusic (Slot AS INTEGER)`

Parameters Slot – Music slot you wish to stop from playing.

Integer 1 – 512	Music slot.
CDMUSIC	CD Audio handle.

Definition Stops a playing music slot.

Comments Will do nothing if no music is playing in this slot.

DS4QB.GetMusicLength& (Function)

Prototype `DECLARE FUNCTION DS4QB.GetMusicLength& (Slot AS INTEGER)`

Parameters Slot – Music slot you wish to get the length from.

Integer 1 – 512	Music slot.

Purpose Retrieves a music slot's length.

Returns Music's length. Length will be in bytes for Mp3 type music, and in order/row for MOD type. (use **GetOrder&** and **GetRow&**)

Comments For a more in-depth tutorial, see: Advanced topics → Music: Advanced topics → Positions.

DS4QB.GetMusicPosition& (Function)

Prototype `DECLARE FUNCTION DS4QB.GetMusicPosition& (Slot AS INTEGER)`

Parameters Slot – Music slot you wish to get the position from.

Integer 1 – 512	Music slot.
CDMUSIC	CD Audio handle.

Purpose Retrieves a music slot's current play position.

Returns Music's current play position. Position will be in bytes for Mp3 type music, seconds for CD Audio, and in order/row for MOD type. (use **GetOrder&** and **GetRow&**)

Comments For a more in-depth tutorial, see: Advanced topics → Music: Advanced topics → Positions.

Combine& (Function)

Prototype `DECLARE FUNCTION Combine& (Order AS LONG, Row AS LONG)`

Parameters Order – Order you wish to combine.

Long Integer 0 – ?	Order.

Row – Row you wish to combine.

Long Integer 0 – ?	Row.

Purpose Combines an order and row into one number, used with **DS4QB.SetMusicPostion** (for MOD type music).

Returns A long integer containing the combined order/row.

Comments For a more in-depth tutorial, see: Advanced topics → Music: Advanced topics → Positions.

GetOrder& (Function)

Prototype `DECLARE FUNCTION GetOrder& (Whole AS LONG)`

Parameters Whole – Whole you wish to retrieve the order from.

Long Integer 0 – ?	Whole.

Purpose Retrieves the order from a whole, the whole is returned from **DS4QB.GetMusicLength&** and **DS4QB.GetMusicPosition&** (for MOD type music).

Returns A long integer containing the order.

Comments For a more in-depth tutorial, see: Advanced topics → Music: Advanced topics → Positions.

GetRow& (Function)

Prototype `DECLARE FUNCTION GetRow& (Whole AS LONG)`

Parameters Whole – Whole you wish to retrieve the row from.

Long Integer 0 – ?	Whole.

Purpose Retrieves the row from a whole, the whole is returned from **DS4QB.GetMusicLength&** and **DS4QB.GetMusicPosition&** (for MOD type music).

Returns A long integer containing the row.

Comments For a more in-depth tutorial, see: Advanced topics → Music: Advanced topics → Positions.

CD Audio Functions

DS4QB.InitCD

Prototype `DECLARE SUB DS4QB.InitCD ()`

Parameters None.

Definition Initializes the CD drive.

Comments If there is no CD in the drive, or any other errors, DS4QB++ (and your program) will continue running.

DS4QB.DeinitCD

Prototype `DECLARE SUB DS4QB.DeinitCD ()`

Parameters None.

Definition Deinitializes the CD drive.

Comments If there is no CD in the drive, or any other errors, DS4QB++ (and your program) will continue running.

DS4QB.PlayCD

Prototype `DECLARE FUNCTION DS4QB.PlayCD (Track AS INTEGER, Looping AS INTEGER)`

Parameters Track – The track number of the song on the CD in which to play.

Integer 1 – ?	Track.

Looping – Whether or not the song will repeat after it is played.

ACTIVE	Turn looping on.
DEACTIVE	Turn looping off.

Definition Plays the specified audio track on the CD.

Comments If there is no CD in the drive, or any other errors, DS4QB++ (and your program) will continue running.

DS4QB.GetCDTracks (Function)

Prototype `DECLARE FUNCTION DS4QB.GetCDTracks ()`

Parameters None.

Purpose Retrieves the number of tracks on the current CD.

Returns An integer 1 or greater containing the track number. Less than 1 if there is no CD in the drive or no audio tracks on the CD.

Comments If there is no CD in the drive, or any other errors, DS4QB++ (and your program) will continue running.

DS4QB.GetCDTrackLength& (Function)

Prototype `DECLARE FUNCTION DS4QB.GetCDTrackLength& (Track)`

Parameters Track – The track number of the song on the CD in which to play.

Integer 1 – (number of tracks)	Track.

Purpose Retrieves the track length (in seconds).

Returns An integer containing the track length.

Comments If there is no CD in the drive, or any other errors, DS4QB++ (and your program) will continue running.

Generic 2D Functions

DS4QB.Set2DPosition

Prototype `DECLARE SUB DS4QB.Set2DPosition (PosX AS INTEGER, PosY AS INTEGER, Angle AS INTEGER)`

Parameters PosX, PosY – Location of the ear.

Integer ? – ?	Sound position.
CURRENT	CURRENT values.

Angle – Angle in which the ear is facing.

Integer 0 – 359	Sound orientation.
CURRENT	CURRENT value.

Definition Sets the CURRENT position/orientation of the ear.

Comments Will only have effect sounds played with the 2D sound routines.

DS4QB.Set2DDistFactor

Prototype `DECLARE SUB DS4QB.Set2DDistFactor (DistF AS SINGLE)`

Parameters DistF – Distance factor.

Single Precision	? – ?	Distance factor.
DEFAULT		Use the DEFAULT setting. (64.0)

Definition Sets the distance factor to be used when calculating the final volume of all sounds played with the 2D audio routines.

Comments For a more in-depth tutorial, see Advanced Topics → Sound: Advanced Topics → 2D Audio.

Other Functions

RawExtract

Prototype `DECLARE SUB RawExtract (RawFile AS STRING, FileIndex AS INTEGER, ExtFile AS STRING)`

Parameters RawFile – File name of the RAW file.

String	? – ?	File name.

FileIndex – The index of the file within the RAW file you wish to retrieve.

Integer	1 – ?	File index.

ExtFile – Name of the file to extract from the RAW file.

String	? – ?	Extract file name.

Definition Extracts a file from a RAW file.

Comments Use this routine with RAW files created by the RAWMAKE utility.

Closing words

Contact information

I can be reached through the following methods:

- ICQ: 58401399
- EMail: lithium@zext.net
- Web: http://lithium.zext.net (post on the message board)

Please contact me if you have any comments, suggestions, bug reports, etc.

Note Please do not complain to me about "lag", as I already know it exists, and am working on correcting it. (Although you shouldn't get much lag, as I have done a lot of work on it.)

Credits

I would like to thank the following people who have made this possible. (I couldn't have done it without you!)

- Ian Luck: BASS.DLL
- Microsoft: DirectSound, MSVC, QuickBasic
- Plasma357: Made .DOC (Word) .PDF and .HTML versions of this document!
- CGI-Joe, DJLauncy, DigitlDud, logiclrd, Sane: Beta testing/ideas
- God: I’m here, aren’t I?

Future Plans

At the moment, I'm not sure… I may or may not go any further with DS4QB++ –To tell you the truth, when I started this project I had no idea it would go this far :] … If there is a major bug to be fixed, or a lot of people really want something added/changed drastically, there may be a DS4QB++ v1.3, otherwise not.

Appendices

Constants

Note Combining flags can be done with the OR operand. Say you wanted to have a sound that uses 8-bit, mono, and software mixing. You would enter this for the sound flags:

```
SND.8BIT OR SND.MONO OR SND.SOFTWARE
```

Init flags

Flag	Purpose
INIT.8BIT	Use 8-bit instead of 16-bit.
INIT.MONO	Use mono instead of stereo.
INIT.3DENABLE	(Reserved for future use.)
INIT.OGGENABLE	Enable OGG Vorbis support.

Sound flags

Flag	Purpose
SND.8BIT	Use 8-bit instead of 16-bit.
SND.MONO	Use mono instead of stereo.
SND.LOOPING	Sound will loop.
SND.3DENABLE	(Reserved for future use.)
SND.SOFTWARE	Use software mixing, not hardware.

Music flags

Flag	Purpose
MUS.RAMP	Use normal ramping.
MUS.RAMPS	Use sensitive ramping.
MUS.FT2MOD	Play MOD as FastTracker 2 would.
MUS.PT1MOD	Play MOD as ProTracker 1 would.
MUS.LOOPING	Music will loop.
MUS.MONO	Use mono instead of stereo.
MUS.SURROUND	Enable surround sound capabilities.
MUS.SURROUND2	Use surround sound mode 2.
MUS.3DENABLE	(Reserved for future use.)
MP3.USEMEM	Load MP3 into memory.
MP3.LOWQUALITY	Play MP3 in low quality.

Constants

Flag	Value
HIGHQUALITY	44,100 Hz
MEDIUMQUALITY	22,050 Hz
LOWQUALITY	11,025 Hz
ACTIVE	1
DEACTIVE	0
WIN9X	1
WINNT	0
NULL	0
CDMUSIC	CD Audio handle, can only be used with some routines.

DEFAULT values

Routine	Attribute	Value
DS4QB.SetGlobalVolumes	SoundVol	50
DS4QB.SetGlobalVolumes	MusicVol	50
DS4QB.SetMasterVolume	Volume	50
DS4QB.Init	SoundQuality	MEDIUMQUALITY
DS4QB.Init	Flags	NULL
DS4QB.LoadSound	Flags	NULL
DS4QB.PlaySoundEx	Freq	Original value
DS4QB.PlaySoundEx	Volume	50
DS4QB.PlaySoundEx	Pan	0
DS4QB.PlaySoundEx	Looping	DEACTIVE
DS4QB.PlaySound2DEx	Freq	Original value
DS4QB.PlaySound2DEx	Volume	50
DS4QB.SetSoundAttr	Freq	Original value
DS4QB.SetSoundAttr	Volume	50
DS4QB.SetSoundAttr	Pan	0
DS4QB.SetSoundAttr	Looping	DEACTIVE
DS4QB.SetSoundAttr	Flags	NULL
DS4QB.AddSound	Freq	Original value
DS4QB.AddSound	Volume	50
DS4QB.AddSound	Pan	0
DS4QB.AddSound	Looping	DEACTIVE
DS4QB.Add2DSound	Freq	Original value
DS4QB.Add2DSound	Volume	50
DS4QB.LoadMusic	Flags	MUS.LOOPING
DS4QB.MusicFadeIn	FStep	5
DS4QB.MusicFadeIn	ObjVol	50
DS4QB.MusicFadeOut	FStep	5
DS4QB.MusicFadeOut	ObjVol	50

(continued on next page)

(continued from previous page)

DS4QB.MusicFadeSwitch	FStep	5
DS4QB.MusicFadeSwitch	ObjVol	50
DS4QB.SetMusicAttr	Volume	50
DS4QB.SetMusicAttr	Pan	50
DS4QB.Set2DDistFactor	DistF	64.0