QBXFMI

AdLib(tm) and SoundBlaster(tm)

Resident FM Music Driver Interface Library

for

QuickBASIC 4.x and BASIC 7.x

by

Cornel Huth

5-May-1991

QBXFMI is an AdLib(tm)/SoundBlaster(tm) compatible resident FM
music sound driver interface module for Microsoft QuickBASIC 4.x
and BASIC 7.x compilers. Its purpose is to provide access to the
resident sound driver for my QBXSBC SoundBlaster/AdLib music card
software library but it can also be used separately.

Features of QBXFMI are:

        1) Linkable module for stand-alone or environment (QLB) use.
        2) Written in QuickBASIC.
        3) Complete interface to all the resident driver functions.

        QBXSBC, the package $19.95
        - includes single-user licenses for:
        1) QBXIOL, fast I/O DOS file module
        2) QBXCTV, digitized voice I/O module for SoundBlaster
        3) QBXFMI, interface module to the resident FM driver for
        the SoundBlaster and AdLib music cards. Includes QBXFMI.BAS
        and resident driver SB-SOUND.COM for the SoundBlaster.
        - also includes:
        4) Useful sample programs in QB demonstrating how to access
        AdLib instrument BNK files, voice files, and card detection.
        5) Programmer documentation

        QBXSBC, assembly source $29.95
        - includes:
        1) QBXIOL.ASM, MASM 5.1 compatible source
        2) QBXCTV.ASM, MASM 5.1 compatible source
        3) Additional programmer documentation

To order see the ORDER.FRM file.

LICENSE AGREEMENT - REGISTERED VERSIONS ONLY

This is a legal agreement between you, the end user, and Cornel Huth. By using this software, you are agreeing to be bound by the terms of this agreement.

SOFTWARE LICENSE

1. GRANT OF LICENSE. Cornel Huth grants to you the right to use one copy of the SOFTWARE on a single terminal connected to a single computer (i.e., with a single CPU). You may not network the SOFTWARE or otherwise use it on more than one computer or computer terminal at a time.

2. COPYRIGHT. The SOFTWARE is owned by Cornel Huth and is protected by United States copyright laws and international treaty provisions. Therefore, you must treat the SOFTWARE like any other copyrighted material (e.g., a book or musical recording) except that you may either (a) make one copy of the SOFTWARE solely for backup or archival purposes, or (b) transfer the SOFTWARE to a single hard disk provided you keep the original solely for backup or archival purposes. You may make a single copy of this document for your own use only.

3. OTHER RESTRICTIONS. You may not rent or lease the SOFTWARE, but you may transfer the software and accompanying documentation on a permanent basis provided you retain no copies and the recipient agrees to the terms of this Agreement. You may not reverse engineer, decompile, or disassemble the software. If the SOFTWARE is an update, any transfer must include the most recent update and all previous versions.

NO WARRANTIES. Cornel Huth disclaims all warranties, either expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the SOFTWARE and the documentation.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. In no event shall Cornel Huth be liable for any damages whatsoever arising out of use of or inability to use this SOFTWARE.

U.S. GOVERNMENT RESTRICTED RIGHTS

SOUND DRIVER DESCRIPTION

The sound driver SB-SOUND.COM is a resident program (TSR) that uses about 6K of code space (13K for the SOUND.COM AdLib driver) and an event queue of from 1-64K. Its purpose is to provide a consistent interface to the music synth card across multiple computer languages. Communication to the driver is through a software interrupt call (INT 65h).

Features of the driver are:

> 1) Event-driven queue for each voice. Basically, what you do is develop a piece of music and write it to the event queue. Notes, volumes, timbre (ADSR) info, timings (when a particular part of your piece is to be played or altered), tempo, all sorts of information, can be put in the queue. When you're done, just tell it to start and away it goes. All processing is done in the background, meaning that you can have your program do something else while the driver PLAYS the piece.

> 2) Variable buffer space. When you load the driver you can specify the size of the event queue buffer with a /Bxx command line parameter.

>   C>SB-SOUND /B64

> Will set aside 64K of RAM for the event queue. To uninstall the driver, use /U. To change the port assignment use /Pxxx. The default buffer space is 4K (valid range is 1-64K) and the default (base) port is 220h. The AdLib SOUND.COM driver is similar but cannot be uninstalled. Also, while you can use the SOUND.COM driver with the SoundBlaster, you cannot use the SB-SOUND.COM driver with an AdLib.

FUNCTION LIST

```
FUNCTION FMInit(Version)
SUB FMSetRelTimeStart(TimeNum,TimeDen)
SUB FMSetState(State)
SUB FMGetState(State)
SUB FMFlush()
SUB FMSetMode(Mode)
SUB FMGetMode(Mode)
FUNCTION FMSetRelVolume(VolNum,VolDen,TimeNum,TimeDen)
FUNCTION FMSetTempo(Tempo,TimeNum, TimeDen)
SUB FMSetKBXpose(Transpose)
SUB FMGetKBXpose(Transpose)
SUB FMSetActVoice(Voice)
SUB FMGetActVoice(Voice)
FUNCTION FMPlayNoteDelay(Pitch,LenNum,LenDen,DelNum,DelDen)
FUNCTION FMPlayNote(Pitch,LenNum,LenDen)
FUNCTION FMSetVoiceTimbre(VTDseg,VTDoff,TimeNum,TimeDen)
FUNCTION FMSetPitchBend(DeltaNum,DeltaDen,TimeNum,TimeDen)
SUB FMSetTickBeat(TickBeat)
SUB FMNoteOn(Voice,Pitch)
SUB FMNoteOff(Voice)
SUB FMSetDirectTimbre(Voice,VTDseg,VTDoff)
SUB FMSetPitchBendRange(Range)
SUB FMSetWaveformParm(OnOff)
FUNCTION FMdriver(Func)
FUNCTION FMdetect(Port)
```

FMInit

Type	FUNCTION - INTEGER
Arguments	Version - INTEGER (returned)
Syntax	stat=FMinit(Version)
Use	Return the software version of the resident sound driver, if installed.
Example 1	stat = FMinit(Version) IF stat THEN PRINT "DRIVER NOT INSTALLED"
Rules	none
Notes	Version 1.51 will be returned as 151 in Version.
Return	0 okay 1 resident sound driver not installed

FMSetRelTimeStart

Type   SUB

Arguments   TimeNum - INTEGER (0-65535)
                TimeDen - INTEGER (1-65535)

Syntax   SetRelTimeStart TimeNum,TimeDen

Use   Set the time origin for all future timing
                references.

Example 1   For example, TimeNum=0 and TimeDen=1 will set the
                time origin at the absolute beginning. Then, if
                you were to FMSetRelVolume(1,2,5,1), the relative
                volume of the then active voice would play at 100%
                volume for beats 1-4 and then decrease 50% for the
                5th and following beats.

Example 2   Let's say that instead of TimeNum=0 and TimeDen=1
                 you used TimeNum=10 and TimeDen=1. Then, if you
                 were to FMSetRelVolume(1,2,5,1),   the relative
                 volume of the then active voice would not be
                 affected. This is because you set the time origin
                 to a point in time after the 5,1 time of the
                 volume change.

Rules   TimeNum can be 0 to 65535, TimeDen 1 to 65535.

Notes   The is a feature of the sound driver, not the
                 music card itself. Try using different settings to
                 get a feel in how to use the timing features.
                 TimeNum/TimeDen are numerator/denominator.

Return   none

FMSetState

Type   SUB

Arguments   State - INTEGER (0-1)

Sytax   FMSetState State

Use   Start, stop, or suspend sound driver output.

Example 1   FMSetState 0   'driver off
                 LoadMusicScore 'go get score, timings, etc

```
            StuffQueue       'give it to the driver
            FMSetState 1     'tell driver to play
```

Rules   none

Notes   You do not need to stop the driver to store data
                to it. However, when just starting, it is best to
                get the driver PRIMED a bit by having a few
                seconds or so of data in it so things flow
                smoothly.

Return   none

FMGetState

Type   SUB

Arguments   State - INTEGER (returned)

```
Sytax          FMGetState State

Use            Determine if the driver is still playing.

Example 1      FMSetState 0      'driver off
               LoadMusicScore  'go get score, timings, etc
               StuffQueue       'give it to the driver
               FMSetState 1      'tell driver to play
               DO                'wait until it's done
                   FMGetState State
               LOOP WHILE State

Rules          none

Notes          State is 1 while music is  still playing, 0 if the
               music is finished or stopped with FMSetState 0.

Return         none
```

FMFlush

Type   SUB

Arguments   none

Syntax   FMFlush

Use   Silence all voices and empty all event queues.

Example 1   PRINT "Press a key to end song and start next"
                DO:LOOP WHILE INKEY$ = ""
                FMFlush
                StartNextSong

Rules   none

Notes   This function could be considered a warm-start
                while FMInit() could be considered a cold-start.

Return   none

FMSetMode

Type   SUB

Arguments   Mode - INTEGER (0-1)

Syntax   FMSetMode Mode

Use   Set the music card to Percussion or Melodic mode.
                Also sets all relative volumes to 100%, all voices
                to piano timbre, and pitch to normal (0 or middle
                C).

Example 1   FMSetMode 1   'set to percussion mode

Rules   none

Notes   In percussion mode (mode=1) voices 0 to 10 are
                available. In melodic (mode=0), 0 to 8.

Return   none

FMGetMode

This function is not curently implemented by either SOUND.COM or
SB-SOUND.COM.

```
FMSetRelVolume

Type            FUNCTION - INTEGER

Arguments       VolNum - INTEGER (0-255)
                VolDen - INTEGER (1-255)
                TimeNum - INTEGER (0-65535)
                TimeDen - INTEGER (1-65535)

Syntax          stat=FMSetRelVolume(VolNum,VolDen,TimeNum,TimeDen)

Use             Change the relative volume of  the active voice at
                the given time. VolNum/VolDen must be less than or
                equal to 1 (where 1 is 100% volume).

Example 1       FMSetRelTimeStart 0,1
                FMSetActVoice 0
                'set relative volume of voice 0 to 50%
                stat = FMSetRelVolume(10,20,0,1)
                'rising 5% after every beat
                stat = FMSetRelVolume(11,20,1,1)
                stat = FMSetRelVolume(12,20,2,1)
                'and so on until beat 10, 100%
                stat = FMSetRelVolume(20,20,10,1)
                'do some other stuffing and then start
                FMSetState 1

Rules           Affects relative volume of the active voice only.

Notes           none

Return          0 okay
                2 queue buffer full
```

```
FMSetTempo

Type            FUNCTION - INTEGER

Arguments       Tempo - INTEGER (0-65535)
                TimeNum - INTEGER (0-65535)
                TimeDen - INTEGER (1-65535)

Syntax          stat=FMSetTempo(Tempo,TimeNum,TimeDen)

Use             Change the tempo to Tempo at the given time.

Example 1       FMSetRelTimeStart 0,1
                'start tempo at 80 beats/min
                stat=FMSetTempo 80,0,1
                'change to double-time at 10th beat
                stat=FMSetTempo 120,10,1

Rules           See FMSetTickBeat()

Notes           none

Return          0 okay
                2 queue buffer full
```

FMSetKBXpose

Type   SUB

Arguments   Transpose - INTEGER (-96-96 max)

Syntax   FMSetKNXpose Transpose

Use   Slide, or transpose, all notes up or down the
                keyboard Transpose number of semitones. Since
                notes can range from -48 to +47, a max Transpose
                range of plus/minus 96 is in order.

Example 1   'play a middle-C whole note
                FMPlayNote 0,4,1
                'wait 'til it's done then
                'play C an octave down
                FMSetKBXpose -12
                'gin and tonic, thank you
                FMPlayNote 0,4,1

Rules   See FMSetTickBeat()

Notes   This function does not work in SB-SOUND.COM. It
                does with SOUND.COM.

Return   0 okay
                2 queue buffer full

FMGetKBXpose

Type   SUB

Arguments   Transpose - INTEGER (returned)

Syntax   FMGetKBXpose Transpose

Use   Get the current transpose value.

Example 1   'play a middle-C whole note
              FMPlayNote 0,4,1
              'wait 'til it's done then
              'play C an octave down
              FMSetKBXpose -12
              'gin and tonic, thank you
              FMPlayNote 0,4,1
              CALL OtherModule
              END

              'where other module can determine transpose
              'OTHER MODULE CODE:
              FMGetKBXpose Transpose

Rules   See FMSetTickBeat()

Notes   This function is not documented by AdLib.

Return   none

FMSetActVoice

Type   SUB

Arguments   Voice - INTEGER (0-8 or 0-10)

Syntax   FMSetActVoice Voice

Use   Set the active voice for all future sound driver
               commands that act on a specific voice. In melodic
               mode, voices 0 to 8 are available. In percussion
               mode, voices 0 to 10.

Example 1   'tell driver we are sending commands for voice 0
               FMSetActVoice 0
               'send some stuff for voice 0
               stat=FMSetVoiceTimbre(vseg,voff,1,0)
               'and so on

Rules   none

Notes   The sound driver assembles everything in its
               buffer by voice and by time. Rather than send all
               voice 0 info and possibly overflowing the buffer
               (it defaults to 4K, max=64K), send enough of each
               voice to be used so that you can FMSetState 1 to a
               smooth start. Thereafter, you can periodically
               refresh the buffer. The driver dynamically
               allocates available buffer space. This means that
               each voice is not assigned a certain percentage of
               the buffer but rather that each voice uses as much
               as it requires. This is why you should avoid
               stuffing an entire voice at a time (unless you
               know that you'll have the buffer space).

Return   none

FMGetActVoice

Type   SUB

Arguments   Voice - INTEGER (returned)

Syntax   FMSetGetVoice Voice

Use   Get the active voice.

Example 1   FMGetActVoice Voice

Rules   none

Notes   This function is not documented by AdLib.

Return   none

FMPlayNoteDelay

Type   FUNCTION - INTEGER

Arguments   Pitch - INTEGER (-48-47)
                LengthNum - INTEGER (0-65535)

```
            LengthDen - INTEGER (1-255)
            DelayNum - INTEGER (0-65535)
            DelayDen - INTEGER (1-255)

Syntax      stat=FMPlayNoteDelay(Pitch,LengthNum,LengthDen,_
                DelayNum,DelayDen)

Use         Play the note of Pitch  of LengthNum/LengthDen and
            set the  delay until  the next  note is  played to
            DelayNum/DelayDen. This note  will play after  the
            previous note's delay is over.

Example 1   'play C whole note, rest half
            FMPlayNote 0,4,1,2,1

Rules       none

Notes       none

Return      0 okay
            2 queue buffer full
```

```
FMPlayNote

Type            FUNCTION - INTEGER

Arguments       Pitch - INTEGER (-48-47)
                LengthNum - INTEGER (0-65535)
                LengthDen - INTEGER (1-255)

Syntax          stat=FMPlayNote(Pitch,LengthNum,LengthDen)

Use             Play the note of Pitch of LengthNum/LengthDen  and
                set the delay also to LengthNum/LengthDen.

Example 1       'play C whole note, rest whole
                FMPlayNote 0,4,1

Rules           none

Notes           none

Return          0 okay
                2 queue buffer full
```

```
FMSetVoiceTimbre

Type          FUNCTION - INTEGER

Arguments     vseg - INTEGER (VARSEG)
              voff - INTEGER (VARPTR)
              TimeNum - INTEGER (0-65535)
              TimeDen - INTEGER (1-65535)

Syntax        stat=FMSetVoiceTimbre(vseg,voff,TimeNum,TimeDen)

Use           Change the timbre (qualities)  of the active voice
              at time TimeNum/TimeDen.

Example 1     'set voice 0 to bongo at time 0
              FMSetActVoice 0
              vseg=VARSEG(bongo(0)):voff=VARPTR(bongo(0))
              stat=FMSetVoiceTimbre(vseg,voff,0,1)

Rules         vseg:voff must point to an  integer data array. In
              other words, each voice attribute is to be 2 bytes
              (eventhough in  BNK files  they are  1 byte).  The
              vseg:voff data must remain valid until the data is
              actually sent to  the hardware,  which will be  at
              TimeNum/TimeDen.  Thereafter,  you can  dispose of
              it.

Notes         See  QBXFMI.BAS  for  an  example  of  using  this
              function. FM sound is  created by the  interaction
              of two operators on the  hardware. The timbre data
              programs these  operators. In  all melodic  voices
              and  percussion voice  6,  the sound  generated is
              from the interaction of the modulator operator and
              the  carrier  operator.  In  the other  percussion
              voices (7-10) only  the modulator operator has  an
              affect on the sound. See QBXFMI.BAS for the format
              of the timbre data.

Return        0 okay
              2 queue buffer full
```

FMSetPitchBend

Type   FUNCTION - INTEGER

Arguments   DeltaNum - INTEGER (-100-100)
                DeltaDen - INTEGER (1-100)
                TimeNum - INTEGER (0-65535)
                TimeDen - INTEGER (1-65535)

Syntax   stat=FMSetPitchBend(100,1,10,1)

Use   Change   the   pitch   at   TimeNum/TimeDen   by
                DeltaNum/DeltaDen where DeltaNum/DeltaDen is from
                -1 to +1 semitones.

Example 1   'change the pitch just a bit (for perfect pitch)
                stat=FMSetPitchBend(1,100,0,1)
                'this changes the pitch up 1/100th a semitone, a
                'very small change indeed

Rules   none

Notes   See FMSetPitchBendRange()

Return   none

FMSetTickBeat

Type   SUB

Arguments   TickBeat - INTEGER (0-65535)

Syntax   FMSetTickBeat

Use   Sets the computer's timer interrupt to allow more
                or less interrupts per unit time.

Example 1   FMSetTickBeat 64

Rules   All notes for all voices should be multiples of
                1/TickBeat. This formula should also remain valid:

                    $18.2 <= (TickBeat * Tempo/60)$

Notes   The interrupts/sec=max(60,TickBeat)*Tempo/60

Return   none

FMNoteOn

Type   SUB

Arguments   Voice - INTEGER (0-8 or 0-10)
                Pitch - INTEGER (-48-47)

Syntax   FMNoteOn(Voice,Pitch)

Use   Directly play the note Pitch using Voice. This
                bypasses the event queue, playing immediately and
                continuously until FMNoteOff().

Example 1   'play C voice 0
                FMNoteOn 0,0

Rules   none

Notes   none

Return   none

FMNoteOff

Type   SUB

Arguments   Voice - INTEGER (0-8 or 0-10)

Syntax   FMNoteOff(Voice)

Use   Turn off the note that was played by FMNoteOn().

Example 1   'play C voice 0 and wait for a keypress
                FMNoteOn 0,0
                SLEEP
                FMNoteOff 0

Rules   none

Notes   none

Return   none

FMSetDirectTimbre

Type   SUB

Arguments   Voice - INTEGER (0-8 or 0-10)
                vseg - INTEGER (VARSEG)
                voff - INTEGER (VARPTR)

Syntax   stat=FMSetDirectTimbre(Voice,vseg,voff)

Use   Directly set the timbre (qualities) of Voice. This
                bypasses the event queue, changing the timbre
                immediately.

Example 1   vseg=VARSEG(bongo(0)):voff=VARPTR(bongo(0))
                stat=FMSetDirectTimbre(0,vseg,voff)
                FMNoteOn 0,0
                SLEEP 1
                FMNoteOff 0

Rules   vseg:voff must point to an integer data array. In
                other words, each voice attribute is to be 2 bytes
                (eventhough in BNK files they are 1 byte). The
                vseg:voff data need not remain valid after the

call.

Notes   No timings are handled by this function. It plays
                when you call it unlike FMPlayNote which plays at
                the time you specifiy.

Return   none

        FMSetPitchBendRange

        Type   SUB

        Arguments   Range - INTEGER (1-12)

        Syntax   FMSetPitchBendRange(Range)

        Use   Alter the pitch-change step of FMSetPitchBend().
                        FMInit() sets   the default   pitch change   of
                        FMSetPitchBend() to 1 semitone. This function
                        changes that default from 1 to 12 semitones.

        Example 1   'change the pitch just a bit (for perfect pitch)
                        stat=FMSetPitchBend(1,100,0,1)
                        'this changes the pitch up 1/100th a semitone, a
                        'very small change indeed
                        FMSetPitchBendRange 12

```
                     stat = FMSetPitchBend(1,100,0,1)
                     'now this changes the pitch up 1/100th an octave
```

Rules   Driver versions 1.3+ only. Version 1.3 has been
                available since November 1988.

Notes   See FMSetPitchBend()

Return   none

FMSetWaveformParm

Type   SUB

Arguments   State - INTEGER (0-1)

Syntax   FMSetWaveformParm(State)

Use   Tells   the   driver   that   there   are   waveform
                parameters in the timbre   data. When State=1
                FMSetVoiceTimbre() and FMSetDirectTimbre() expect
                an array of 28 integers where the last two are the

```
                waveform  operators.  State=0 tells  it  to expect
                only 26  (having  28  will  not  affect  it,  the
                waveform parms simply won't be used).

Example 1       'tell it we have WF parms
                FMSetWaveformParm 1
                vseg=VARSEG(bongo(0)):voff=VARPTR(bongo(0))
                stat=FMSetDirectTimbre(0,vseg,voff)
                FMNoteOn 0,0
                SLEEP 1
                FMNoteOff 0

Rules           none

Notes           FMInit() sets this  to 0. To enable it,  call this
                function after FMInit().

Return          none
```

```
NOTE DATA

Notes are made up of 3 components: pitch, duration, and delay.
PITCH
                      261.63Hz
     C    -48  -36  -24  -12 *0*   12   24   36
```

```
             C#                          1 -277.18
             D                           2 -293.66
             D#                          3 -311.13
             E                           4 -329.63
             F                           5 -349.23
             F#                          6 -369.99
             G                           7 -391.99
             G#                          8 -415.31
             A                           9 -440.00
             A#                          10-466.16
             B     -37  -25  -13  -1  11   23   35   47
                                     493.88Hz
```

The 0 represents the note of middle C on the piano keyboard and
is equal to 261.63Hz. Columns to the right of 0 are an octave
above middle C, to the left, an octave below. To calculate the
frequency of a note, use the frequencies listed by each note and
double it for each octave above or halve it for each below. The
lowest frequency is 16.352Hz (pitch=-48), the highest is 3951.1
Hz (pitch=47), not including harmonics.

DURATION

The length of a note is given as a numerator over denominator.
The effective duration of a note is a function of the tempo.

Name	Num	/ Den
Whole note	4	1
Dotted half	3	1
Half note	2	1
Dotted quarter	3	2
Quarter note	1	1
Dotted eighth	3	4
Eighth note	1	2
Dotted sixteenth	3	8
Sixteenth	1	4

DELAY

Delay is the time after the current note starts playing that the
next should be started. Usually, delay equals note duration so
that the next note plays immediately after the current note ends.
If there is a further delay, that delay is called at rest. Rests
can also be 'played' alone by specifying a note of duration 0
with the required delay.

TIMBRE DATA

The timbre data is used to program each of the two operators (18 total) that are used to create FM sound. In melodic voices 0 to 8, there are two operators used together to create a sound. In percussion voices 7-10, 1 operator is used to create the sound. You can select from two modes: melodic only or melodic-percussive. In melodic only you have voices 0-8 available. In melodic-percussive, voice 0-5 are melodic, percussion voice 6 (Bass Drum using 2 operators), percussion 7,8,9,10 are the Snare, TomTom, Cymbal, and Hi-Hat, respectively (each using 1 operator).

The timbre data parameters are made up of components: envelope parms, oscillator parms, and level controller parms. Each of the two operators have 13 parms (0-12) plus each has an additional waveform parameter.

ENVELOPE

ADSR - Attack Rate, Decay Rate, Sustain Level, Release Rate. These parms tell how quickly a sound starts (attack rate), how quickly it falls (decay rate) to the sustain level (sustain level) and, once the note has been released, how quickly it decays to zero-level (release rate).

    AR 0-15 (0=slow attack) (Timbre parm 3)
    DR 0-15 (0=slow decay) (6)
    SL 0-15 (0=maximum sustain level) (4)
    RR 0-15 (0=slow release) (7)

Other envelope parms are Sustaining Sound and Envelope Scaling. Sustaining sound directs whether the sustain level will be held or if the release should begin immediately after reaching the sustain level. Envelope Scaling tries to adjust the envelope parameters to more accurately mimic musical instruments.

    SS 0-1 (1=on,0=off) (5)
    ES 0-1 (1=on,0=off) (11)

OSCILLATOR

Frequency Multiplier, Frequency Vibrato, Modulation Feedback. The frequency multiplier lets you modify the oscillator frequency so that a sound becomes a multiple of the original note, or harmonic.

MULTI (1)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
factor	.5	1	2	3	4	5	6	7	8	9	10	10	12	12	15

Frequency vibrato creates an automatic variation to the oscillator's frequency (very slight). Also called Pitch Vibrato.

        VIB 0-1 (1=7 cents frequency fluctuation depth,0=none) (10)
                (there are 1200 cents in an octave)

Modulation feedback sets the modulator's oscillator to feedback its output back to its input. (Not used by the carrier op.)

        FB (2)   0   1   2   3   4   5   6   7
        modulation   0   p/16 p/8 p/4 p/2 p   2p   4p
        (p=PIE)

LEVEL

Output Level, Level Scaling, Amplitude Vibrato. These adjust the overall output of each operator.

Output level adjusts the operator's maximum output. The modulator output level determines the intensity of the modulation if the carrier and the carrier output level determines the overall volume of the sound.

        OL 0-63 (0=max,63=min) (8)
                (to convert to dB: dB=OL*.75)

Level scaling, or Key Scale Level, adjusts the output level of higher notes so that they play less loud than lower notes.

        KSL (0)   0 1   2   3
        dB/octave 0 3   1.5 6
        drop

Amplitude vibrato creates an automatic variation to the operators output level. Also called tremolo.

        AM 0-1 (1=1dB depth of fluctuation,0=none) (9)

Another parameter is the Frequency Modulation/Additive flag. Valid only for the modulator operator, is changes the way the sound is created.

        FM 0-1 (0=use FM,1=use additive synthesis) (12)