# DxMidi v2.0

*Reference manual*

Realvision Multimedia software

# DxMidi v2.0 - Implementation

## Availability

Boolean = DxMidiMMAvail ()
Boolean = DxMidiOMSAvail ()

## Sign in / Sign out

Integer = DxMMconnect( string , string )
Integer = DxMMdisconnect ()
Integer = DxOMSconnect ( string , string )
Integer = DxOMSdisconnect ()

## Setting buffers

Integer = DxSetBufferNotes ( integer )
Integer = DxSetBufferSysex ( integer )

## Sending events

Integer = DxNoteON ( integer , integer , integer )
Integer = DxNoteOFF ( integer , integer )
Integer = DxSendData ( string , integer )
Integer = DxDurNote ( integer , integer , integer , integer )
Integer = DxSendControl ( integer , integer , integer )

## Receiving events

Boolean = DxDataFlush ()
Integer = DxIsDatas ()
string = DxReceiveData ()
Boolean = DxMidiDataFilter ( Boolean , Boolean , Boolean )

## OMS suppor t

Boolean = DxOMSStudioSetup ()
Boolean = DxOMSMidiSetup ()
Boolean = DxOMSFilterIN ( integer )
Boolean = DxOMSFilterOUT ( integer )
Integer = DxOMSGetNodes ( integer )
Integer = DxOMSPlugOutNode ( integer )
Integer = DxOMSPlugInNode ( integer )
string = DxOMSGetNodeOUTName ( integer )
string = DxOMSGetNodeINName ( integer )
Integer = DxOMSDisconnectINNode ( integer )
Integer = DxOMSNodesINDial ( String, Boolean )
Integer = DxOMSGetNodeINUniqueID ( Integer )
Integer = DxOMSGetNodeOUTUniqueID ( Integer )
Integer = DxOMSPlugOutUniqueID ( Integer )
Integer = DxOMSPlugInUniqueID ( Integer )
Boolean = DxOMSWorldChanged ()

**SysEx suppor t**
Boolean = DxStartSysEx ( Integer , Integer )
Boolean = DxIsSysEx ()
String = DxReadSysEx ()
Boolean = DxStopSysEx ()

**Midi Sequencer support (V2 pro only)**
Boolean = DxOpenSequencer ( Integer )
Boolean = DxSeqSetTrackChannel ( Integer , Integer )
Integer = DxMidiSeqRecTrack ( Integer )
Boolean = DxMidiSeqClearTrack ( Integer )
Boolean = DxMidiSeqMuteTrack ( Integer , Boolean )
Integer = DxMidiSeqKillRec ()
Boolean = DxCloseSequencer ()
Integer = DxMidiSeqGetTime ()
Boolean = DxMidiSeqSetTime ( Integer , Integer )
Boolean = DxMidiSeqPlay ()
Boolean = DxMidiSeqTempo ( Integer )
Boolean = DxMidiSeqMMSetting ( Integer , Integer , Integer )
Boolean = DxMidiSeqMMAction ( Boolean )
Integer = DxMidiSeqNote ( Integer , Integer , Integer , Integer , Integer , Integer )
Boolean = DxMidiSeqMTAction ( Boolean )
Boolean = DxMidiMTClear ()
Boolean = DxMidiMTDelEvent ()
Boolean = DxMidiMTGetSize ()
Integer = DxMidiMTGetInfoTime ( Integer )
Boolean = DxMidiMTSetInfoTempo ( Integer , Integer )
Integer = DxMidiMTGetInfoTempo ( Integer )
Boolean = DxMidiMTInsert ( Integer , Integer , Integer )
Integer = DxMidiSeqEvent ( Integer , Integer , Integer , Integer , Integer , Integer )
String = DxMidiGetTrack ( Integer )
Boolean = DxMidiSeqStop ()
String = DxMidiGetTrack ( Integer )
Integer = DxMidiGetTrackSize ( Integer )
Boolean = DxMidiSetTrack ( Integer , String , Integer )
Boolean = DxMidiPaintTrack ( Integer , Integer , Integer , Integer , Integer , Integer , Integer )

## Control methods

Integer = *control.*Note ( Integer , Integer )
Integer = *control.*NoteON ( Integer , Integer , Integer )
Integer = *control.*NoteOF ( Integer , Integer , Integer )
Integer = *control.*DurNote ( Integer , Integer , Integer , Integer )
Integer = *control.*SetBufferNotes ( Integer )
Integer = *control.*SetBufferSysex ( Integer )
Boolean = *control.*StartSysEx ( Integer , Integer )
Boolean = *control.*IsSysEx ()
String = *control.*ReadSysEx ()
Boolean = *control.*StopSysEx ()
Integer = *control.*SendControl ( Integer , Integer , Integer )
Integer = *control.*IsDatas ()
Integer = *control.*SendData ( String , Integer )
Boolean = *control.*Flush ()
String = *control.*ReceiveData ()

# 1/ Availability

*Before writing of reading any midi data, and starting an application, you must call one of these methods to test the disponibility of Midi drivers.*
*These, are global methods that you can place in your program in a APP class, in OPEN and CLOSE events.*
*Don't forget to keep a global variable to store the avail respond for Midi disponibility.*

*Never call other DxMidi methods before one of these, or when **DxMidiMMAvail()** or **DxMidiOMSAvail** replies false.*

## avail = **DxMidiMMAvail** ()

avail as Boolean

Test the Apple Midi Manager v2 availability. If you attempt to connect to Midi Manager, you must test this before any other command to the Midi Manager plugin interface with an hypothetical driver !

If avail is true, then the Midi Manager is available.

## avail = **DxMidiOMSAvail** ()

avail as Boolean

Test the Opcode Music System availability. You must test this before any other command to the plugin interface OMS midi methods !

if avail = true, then OMS is available.

# 2/ Sign IN / OUT

result = DxMMconnect ( osType , name )
osType as string
name as string
result as Integer

Signing for the Midi Manager Interface. The signature is effective only if the Client ID and the Client Name are unique.
Sign before using any other specific Midi Manager method.

Use this method just after a call to *DxMidiMMAvail()*.

You must close the program with a *MMdisconnect()*, or you could keep the old instance of your client ID and name in the MidiManager Patch the next time you sign in.
You can connect to midi at the start of your app, and then close connection at end, even if you are not sure that the user will use the Midi driver.
**osType** is a 4 character code (of your program, to simplify) of the Midi signature connection. It must be unique. Contact Apple to get an unique OStype value for your program and apply this for the connection client ID.
**name** is the name of your program. In the Midi Manager, this name appears in the Patchbay program when your program is launched and a connection made.
The **result** code contains 0 if the connection is made, or a standard DxMidi result code for error.

***example code :***

```
// the control name chosen is : midi

// create the property
dim p_MMavail as Boolean

// edit me.Open() event
Sub Open()
dim avail as Boolean
dim result as Integer

avail = DxMidiMMAvail
if avail = true then
        // use a property to maintain the MM avail for closing at end
        p_MMavail = avail
        // try to connect and auto-patch
        result = DxMMconnect («abcd», «Midi-Test»)
        if result >= 0 then
                // connection made
                // don't forget to disconnect at end in close() method
                // ... put here some code to play notes
                // but you can put this text anywhere in your program
                end if
        end if
End Sub

Sub Close()

dim avail as Boolean
dim result as Integer

if p_MMavail = true then
        result = DxMMdisconnect()
        // result code tell you if the disconnection is ok
        p_MMavail = false
end if
End Sub
```

## result = **DxMMdisconnect** ()

result as Integer

Use this method to disconnect the connections in the patchbay, and sign out to Midi Manager. Don't forget to do it before quitting the application runtime (see *MMconnect()* example).

The result code contains 0 if the connection is made, or a standard DxMidi result code for error.

## result = **DxOMSconnect** ( osType, name )

osType as String
name as String
result as Integer

Connect to OMS. You must disconnect at end the OMS signin, by an *OMSdisconnect()* method. If you don't do it, you could generates an internal error into OMS. Result is <> 0 if there is an error. See the error codes list at the end of this manual. You can use the same procedure as MidiManager to sign in the OMS studio.

**osType** is a 4 character code (of your program, to simplify) of the Midi signature connection. It must be unique. Contact Apple to get an unique OStype value for your program and apply this for the connection client ID. **name** is the name of your program. In OMS, this name is used as a virtual node to other clients.
The **result** code contains 0 if the connection is made, or a standard DxMidi result code for error.

## result = **DxOMSdisconnect** ()

result as Integer

Disconnect to OMS. This generates a signout applet to the OMS studio to disconnect the driver. None of the OMS methods should be used after this except *DxMidiOMSAvail()* or *OMSConnect()*. For result codes, see the error list codes at the end of this manual.

The result code contains 0 if the connection is made, or a standard DxMidi result code for error.

# 3/ DxMidi - Setting buffers

Because the transfert of Midi events in RealBasic is not fast enough to guarantee all datas transmission, DxMidi manages two buffers. One for the notes (and Sysex too in standard mode), and one another for special SysEx purposes.
This second buffer expects a particular system message to occur in the DxMidi driver and then put all the datas in the Sysex buffer to protect datas from any timeout troubleshooting.
By setting the Notes buffer, you can enlarge the default size of 4000 bytes to every value according to the available memory in the Macintosh (let the user decides for the buffer size himself if you want more configuration settings).
The SysexBuffer is 4000 bytes by default, but to contains large sysex, you can enlarge his size to a size larger than the expected sysex.
You can specify the size of the buffers at every moment, enlarge them, reduce them, and also set the 0 value (but you will never receive datas anymore !).

DxMidi V2 note : the buffers methods are now globals, and you don't use Midi control anymore... don't forget to change this setting in you existing DxMidi V1 project...

**result** = **DxSetBufferNotes(** **size** )
**result** = *control*.SetBufferNotes ( **size** )
size as integer
result as integer

This sets the new value of the notes buffer.
Result returns (-1) if the reservation failed (try to reserve a smaller size). Default value is 4000. Don't try to specify a too large value (>1Mo).

size is the number of bytes reserved
The result code contains 0 if the connection is made, or a standard DxMidi result code for error.

**result = DxSetBufferSysex( size )**
**result =** *control.***SetBufferSysex( size )**
size as integer
result as integer

This sets the new value of the notes buffer. Result gives you (-1) if the reservation failed (try to reserve a smaller size). Default value is 4000. Don’t try to specify a too large value (>1Mo).

size is the number of bytes reserved
The result code contains 0 if the connection is made, or a standard DxMidi result code for error.

# 4/ DxMidi - Sending events

**result** = **DxNoteON** ( **channel** , **note** , **velocity** )
**result** = *control.***NoteON** ( **channel** , **note** , **velocity** )
result as Integer
channel as Integer
note as Integer
velocity as Integer

Plays a note ON data command *(3 datas are sent to Midi: status, note and velocity)*. result code <> 0 if an error occured.
channel is the specified MIDI channel (see MIDI documentation for more infos).
note is the MIDI note (1-127) defined in the MIDI norm.
velocity is the MIDI standard velocity (1-127) defined in the MIDI norm.

Don't overflow any value of the method, either you could have an transmit error.
*NOTE : The MIDI note is sent immediately.*

**result** = **DxNoteOFF** ( **channel** , **note** )
**result** = *control.***NoteOFF** ( **channel** , **note** )
result as Integer
channel as Integer
note as Integer

Plays a note OFF data command (3 datas posted to Midi, status, note and velocity = 0).
In fact, this is always a Note ON command, but with velocity = 0.
result code <> 0 if an error occured.

channel is the specified MIDI channel (see MIDI documentation for more infos).
note is the MIDI note (1-127) defined in the MIDI norm.
The standard MIDI defined a Note OFF velocity, but in most of cases, this parameter is absolutely useless.

**result = DxSendData ( myString , mode )**
**result =** *control.***SendData ( myString , mode )**
result as Integer
myString as String
mode as Integer

This command is useful to send SysEx and controls. The method will send the packet with a mode to autorize sending more than 248 midi packets (for SysEx).
result code <> 0 if an error occured.
mode is the flag for : starting packet, end packet, continued packet, packet to be continued.
You can use this to send notes, but it’s better to use the NoteON method. You can send Controls, Program changes, short and long SysEx events.
myString is a RB-Waste string (larger than 32k), it contains the datas to transmit.
If you want to send a SysEx longer than 248 bytes, send multi-packets and set the mode parameter to inform the Midi driver that the datas are not finished.

this is the values for mode :

| | | |
|---|---|---|
| NoCont | 0 | This first packet is not continued. |
| StartCont | 1 | This is the first packet of a multi-packet message |
| MidCont | 3 | This packet is the following of a long message, and not the end |
| EndCont | 2 | This packet is the following of a long message, and this is the end ! |

Don’t use NoCont to terminate a long multi-packet message.
NoCont is just the default.

**result = DxDurNote ( channel , note , velocity , duration )**
**result =** *control.***DurNote ( channel , note , velocity , duration )**
result as Integer
channel as Integer
note as Integer
velocity as Integer
duration as Integer

Synchronous version only,
to send notes with duration (1/60 of a second).
This method will be deprecated in the future to be replaced by a real-time sequencer.
result code <> 0 if an error occured.
channel is the specified MIDI channel

(see MIDI documentation for more infos).
note is the MIDI note (0-127) defined in the MIDI norm.
velocity is a velocity MIDI value (0-127)

result = **DxSendControl** ( chan , control , value )
result = *control*.**SendControl** ( chan , control , value )
result as Integer
chan as Integer
control as Integer
value as Integer

This control of global method is used to send directly controls like sustain or damper actions to Midi driver. The control setting is defined by MIDI norm as a specific control code. The value is always a setting between 0 and 127.

Control definition list :

| 2nd Byte | Value | | Function | 3rd Byte |
|---|---|---|---|---|
| Binary | Hex | Dec | | Value |
| 00000000 | 00 | 0 | Bank Select | 0-127 |
| 00000001 | 01 | 1 | Modulation wheel | 0-127 |
| 00000010 | 02 | 2 | Breath control | 0-127 |
| 00000011 | 03 | 3 | Undefined | 0-127 |
| 00000100 | 04 | 4 | Foot controller | 0-127 |
| 00000101 | 05 | 5 | Portamento time | 0-127 |
| 00000110 | 06 | 6 | Data Entry | 0-127 |
| 00000111 | 07 | 7 | Channel Volume (formerly Main Volume) | 0-127 |
| 00001000 | 08 | 8 | Balance | 0-127 |
| 00001001 | 09 | 9 | Undefined | 0-127 |
| 00001010 | 0A | 10 | Pan | 0-127 |
| 00001011 | 0B | 11 | Expression Controller | 0-127 |
| 00001100 | 0C | 12 | Effect control 1 | 0-127 |
| 00001101 | 0D | 13 | Effect control 2 | 0-127 |
| 00001110 | 0E | 14 | Undefined | 0-127 |
| 00001111 | 0F | 15 | Undefined | 0-127 |
| 00010000 | 10 | 16 | General Purpose Controller #1 | 0-127 |
| 00010001 | 11 | 17 | General Purpose Controller #2 | 0-127 |
| 00010010 | 12 | 18 | General Purpose Controller #3 | 0-127 |
| 00010011 | 13 | 19 | General Purpose Controller #4 | 0-127 |
| 00010100 | 14 | 20 | Undefined | 0-127 |
| 00010101 | 15 | 21 | Undefined | 0-127 |
| 00010110 | 16 | 22 | Undefined | 0-127 |
| 00010111 | 17 | 23 | Undefined | 0-127 |
| 00011000 | 18 | 24 | Undefined | 0-127 |
| 00011001 | 19 | 25 | Undefined | 0-127 |
| 00011010 | 1A | 26 | Undefined | 0-127 |
| 00011011 | 1B | 27 | Undefined | 0-127 |
| 00011100 | 1C | 28 | Undefined | 0-127 |

| | | | | |
|---|---|---|---|---|
| 00011101 | 1D | 29 | Undefined | 0-127 |
| 00011110 | 1E | 30 | Undefined | 0-127 |
| 00011111 | 1F | 31 | Undefined | 0-127 |
| 00100000 | 20 | 32 | Bank Select | 0-127 |
| 00100001 | 21 | 33 | Modulation wheel | 0-127 |
| 00100010 | 22 | 34 | Breath control | 0-127 |
| 00100011 | 23 | 35 | Undefined | 0-127 |
| 00100100 | 24 | 36 | Foot controller | 0-127 |
| 00100101 | 25 | 37 | Portamento time | 0-127 |
| 00100110 | 26 | 38 | Data entry | 0-127 |
| 00100111 | 27 | 39 | Channel Volume (formerly Main Volume) | 0-127 |
| 00101000 | 28 | 40 | Balance | 0-127 |
| 00101001 | 29 | 41 | Undefined | 0-127 |
| 00101010 | 2A | 42 | Pan | 0-127 |
| 00101011 | 2B | 43 | Expression Controller | 0-127 |
| 00101100 | 2C | 44 | Effect control 1 | 0-127 |
| 00101101 | 2D | 45 | Effect control 2 | 0-127 |
| 00101110 | 2E | 46 | Undefined | 0-127 |
| 00101111 | 2F | 47 | Undefined | 0-127 |
| 00110000 | 30 | 48 | General Purpose Controller #1 | 0-127 |
| 00110001 | 31 | 49 | General Purpose Controller #2 | 0-127 |
| 00110010 | 32 | 50 | General Purpose Controller #3 | 0-127 |
| 00110011 | 33 | 51 | General Purpose Controller #4 | 0-127 |
| 00110100 | 34 | 52 | Undefined | 0-127 |
| 00110101 | 35 | 53 | Undefined | 0-127 |
| 00110110 | 36 | 54 | Undefined | 0-127 |
| 00110111 | 37 | 55 | Undefined | 0-127 |
| 00111000 | 38 | 56 | Undefined | 0-127 |
| 00111001 | 39 | 57 | Undefined | 0-127 |
| 00111010 | 3A | 58 | Undefined | 0-127 |
| 00111011 | 3B | 59 | Undefined | 0-127 |
| 00111100 | 3C | 60 | Undefined | 0-127 |
| 00111101 | 3D | 61 | Undefined | 0-127 |
| 00111110 | 3E | 62 | Undefined | 0-127 |
| 00111111 | 3F | 63 | Undefined | 0-127 |
| 01000000 | 40 | 64 | Damper pedal on/off (Sustain) | <63=off |
| 01000001 | 41 | 65 | Portamento on/off | <63=off |
| 01000010 | 42 | 66 | Sustenuto on/off | <63=off |
| 01000011 | 43 | 67 | Soft pedal on/off | <63=off |
| 01000100 | 44 | 68 | Legato Footswitch | <63=off |
| 01000101 | 45 | 69 | Hold 2 | <63=off |
| 01000110 | 46 | 70 | Sound Controller 1 (Sound Variation) | 0-127 |
| 01000111 | 47 | 71 | Sound Controller 2 (Timbre) | 0-127 |
| 01001000 | 48 | 72 | Sound Controller 3 (Release Time) | 0-127 |
| 01001001 | 49 | 73 | Sound Controller 4 (Attack Time) | 0-127 |
| 01001010 | 4A | 74 | Sound Controller 5 (Brightness) | 0-127 |
| 01001011 | 4B | 75 | Sound Controller 6 | 0-127 |

| | | | | |
|---|---|---|---|---|
| 01001100 | 4C | 76 | Sound Controller 7 | 0-127 |
| 01001101 | 4D | 77 | Sound Controller 8 | 0-127 |
| 01001110 | 4E | 78 | Sound Controller 9 | 0-127 |
| 01001111 | 4F | 79 | Sound Controller 10 | 0-127 |
| 01010000 | 50 | 80 | General Purpose Controller #5 | 0-127 |
| 01010001 | 51 | 81 | General Purpose Controller #6 | 0-127 |
| 01010010 | 52 | 82 | General Purpose Controller #7 | 0-127 |
| 01010011 | 53 | 83 | General Purpose Controller #8 | 0-127 |
| 01010100 | 54 | 84 | Portamento Control | 0-127 |
| 01010101 | 55 | 85 | Undefined | 0-127 |
| 01010110 | 56 | 86 | Undefined | 0-127 |
| 01010111 | 57 | 87 | Undefined | 0-127 |
| 01011000 | 58 | 88 | Undefined | 0-127 |
| 01011001 | 59 | 89 | Undefined | 0-127 |
| 01011010 | 5A | 90 | Undefined | 0-127 |
| 01011011 | 5B | 91 | Effects 1 Depth | 0-127 |
| 01011100 | 5C | 92 | Effects 2 Depth | 0-127 |
| 01011101 | 5D | 93 | Effects 3 Depth | 0-127 |
| 01011110 | 5E | 94 | Effects 4 Depth | 0-127 |
| 01011111 | 5F | 95 | Effects 5 Depth | 0-127 |
| 01100000 | 60 | 96 | Data entry +1 | N/A |
| 01100001 | 61 | 97 | Data entry -1 | N/A |
| 01100010 | 62 | 98 | Non-Registered Parameter Number LSB | 0-127 |
| 01100011 | 63 | 99 | Non-Registered Parameter Number MSB | 0-127 |
| 01100100 | 64 | 100 | Registered Parameter Number LSB | 0-127 |
| 01100101 | 65 | 101 | Registered Parameter Number MSB | 0-127 |
| 01100110 | 66 | 102 | Undefined | ? |
| 01100111 | 67 | 103 | Undefined | ? |
| 01101000 | 68 | 104 | Undefined | ? |
| 01101001 | 69 | 105 | Undefined | ? |
| 01101010 | 6A | 106 | Undefined | ? |
| 01101011 | 6B | 107 | Undefined | ? |
| 01101100 | 6C | 108 | Undefined | ? |
| 01101101 | 6D | 109 | Undefined | ? |
| 01101110 | 6E | 110 | Undefined | ? |
| 01101111 | 6F | 111 | Undefined | ? |
| 01110000 | 70 | 112 | Undefined | ? |
| 01110001 | 71 | 113 | Undefined | ? |
| 01110010 | 72 | 114 | Undefined | ? |
| 01110011 | 73 | 115 | Undefined | ? |
| 01110100 | 74 | 116 | Undefined | ? |
| 01110101 | 75 | 117 | Undefined | ? |
| 01110110 | 76 | 118 | Undefined | ? |
| 01110111 | 77 | 119 | Undefined | ? |
| 01111000 | 78 | 120 | All Sound Off | 0 |
| 01111001 | 79 | 121 | Reset All Controllers | 0 |
| 01111010 | 7A | 122 | Local control on/off | 0=off 127=on |
| 01111011 | 7B | 123 | All notes off | 0 |

| | | | |
|---|---|---|---|
| 01111100 | 7C | 124 | Omni mode off (+ all notes off) 0 |
| 01111101 | 7D | 125 | Omni mode on (+ all notes off) 0 |
| 01111110 | 7E | 126 | Poly mode on/off (+ all notes off) ** |
| 01111111 | 7F | 127 | Poly mode on (incl mono=off +all notes off) 0 |

# 5/ DxMidi - Receiving events

When you connect the INPUT port of DxMidi, and valid the patch connection, you are declared able to receive datas and the Midi In port of the driver is keeping the datas in the limits of its reserved memory (32k). When the memory is out, an automatic procedure deletes the oldest datas and creates place for new ones. We encourage you to read all packets when the connection is active, make periodic tasks to the routine *ReceiveDatas()* or *Flush(),* either, you are about to lose time processing with read data you don't care at all !
On the other hand, if you want to receive Sysex datas for storage, you require no missing datas, because when receiving big flows of data, your Realbasic application get too much time to read the incoming datas (and analyze them), and you will lose important bytes. To prevent yourself from this kind of critical problem, take control of your user behaviour and get the datas only at end of reception, when you are sure that all datas are in the 32k buffer.

**Err** = **DxDataFlush** ()
**Err** = *control.***Flush** ()
Err as Boolean

By this command, you clear the data queue created for reception. This method could be called in an temporary idle for reading and loosing datas. Don't call this when you are not sure if notes are playing, because there will be times when you want to stop the notes on your Midi device.
Err returns an error "False" if the flush failed.

**result** = **DxisDatas** ()
**result** = *control.***isDatas** ()
result as Integer

Tests if datas are incoming to the plugin driver queue. result returns the length of the first data available in the queue. If result = 0, then no data is available.
NOTE : Buffer can be full if you don't read infos in the queue. In these cases, the auto-clear procedure is actived and you will lose datas !

**myString** = **DxReceiveData** ()
**myString** = *control.***ReceiveData** ()
myString as String

Use this method after *IsDatas()* to load datas into myString.
For larger datas (eg. SysEx or controls), read the «info» property sent.
Keep in mind that the packet is cleared from the data queue after this command (but you let some place to following packets).

**Err** = **DxMidiDataFilter** ( **SysExFlag** , **ControlFlag** , **NoteFlag** )
Err as Boolean
SysExFlag as Boolean
ControlFlag as Boolean
NoteFlag as Boolean

This is the read filter function. Use this when you don't want to recei-ve types of datas.

# 6/ DxMidi - OMS suppor t

OMS connection system is slightly different than Midi manager. You work with patchbay in the Midi Manager, or you work with «Nodes» into OMS.
The node is either a real input or output (to or from a Midi interface), or a virtual device (software connection to patch, drive data...).
These nodes are managed by OMS as low level, and you can keep control of these connections to patch into them.
DxMidi plugin permits you to connect one IN and one OUT to the OMS MIDI studio for the global DxMidi object.
Call first the *DxOMSconnect()* method to sign-in, next you can either see the studio, the connections and devices connections made by OMS (call *OMSStudioSetup()* or *OMSMidiSetup()),* or get the nodes for IN and OUT sockets to select :

- one or more connections to other nodes
(eg. real interface IN, virtual INs).

- one connection to the output node
(eg. real interface OUT, QuickTime Instruments, etc.).

The DxMidi Patcher in one-Out-connected to simplify the time-access of your app.

To add rich selection for patching studio, OMS provide properties to filter types of nodes.
See interface constants to add themselves for Node-filtration.

## Err = DxOMSStudioSetup ()

Err as Boolean

This function activates a high-level function *(OMSOpenCurrentStudioSetup()* in the OMS sdk).
OMS recommends that all OMS-compatible applications contain a menu command, Open Current OMS Setup, or OMS Studio Setup..., located immediately before OMS MIDI Setup...
It would be wonderful to add yours in a big final release for «courteous» OMS compatibility.

Err is False if the call generated an error. Never call the function if you are not sure that an OMS connexion is active, this could give you an impredicable arror.

### **Err** = **DxOMSMidiSetup** ()

Err as Boolean

Final commercial applications may have a MIDI Setup... (or OMS MIDI Setup...) menu command. This function activates a high-level function *(OMSMIDISetupDialog()* in the OMS sdk).

Err is False if the call generated an error. Never call the function if you are not sure that an OMS connexion is active, this could give you an impredicable arror.

### **Err** = **DxOMSFilterIN** ( **filters** )

Err as Boolean
filters as Integer

*OMSFilterIN* sets the node-visibility of the MIDI input socket of your DxMidi connection. You can use different types of constants and add them to create the mode value.

Example :

To see the inputs, real inputs and virtual inputs, set filters:

```
dim omsIncludeInputs, omsIncludeReal , omsIncludeVirtualas integer
omsIncludeInputs = 1
omsIncludeReal = 4
omsIncludeVirtual = 8
filters = omsIncludeInputs + omsIncludeReal + omsIncludeVirtual

filters = 1 + 4 + 8 = 13).

interface constants :
omsIncludeInputs = 1
omsIncludeOutputs = 2
omsIncludeReal = 4
omsIncludeVirtual = 8
omsIncludeSync = 16
omsIncludeSyncOnly = 32
omsMergeOutputsByPort = 64
omsIncludeSecret = 128
```

You could define all these constants as properties in a class (may be reusable?) and use these constants to implement your code.

Err returns FALSE if an error occured

### **Err = DxOMSFilterOUT ( filters )**

Err as Boolean
filters as Integer

Like *OMSFilterIN, OMSFilterOUT* sets the node-visibility of the MIDI Output socket of your DxMidi connection. You can use different types of constants filters and add them to create the mode value. See interface constants.

Err returns FALSE if an error occured

### **result = DxOMSGetNodes ( mode )**

result as Integer
mode as Integer

To send or receive datas, you must first call nodes to patch your app. The *OMSGetNodes,* get information about all available nodes for IN and OUT. Set the filter property before calling method (See *DxOMSFilterIN(),*
*DxOMSFilterOUT()).*
For Output nodes, set mode = 0, For inputs, set mode = 1.
The result is the number of nodes found. The nodes list is kept and stay valid in memory from now until a *DxOMSWorldChanged()* detects modifications in the OMS studio. You are limited to 64 nodes inputs/outputs and can access to nodes by number by the following functions.
Result returns a standard DxMidi error code.

### **result = DxOMSPlugOutNode ( node )**

result as Integer
node as Integer

This connect your OUT OMS MIDI socket to a node specified. Note that the socket can only be connected to ONE NODE at time. But it's easy to change node attribution because this method is only a choice, and not a real connection made. You can acces to *DxOMSPlugOutNode()* anytime and anywhere you like in your program, but remember that you must call *DxOMSGetNodes(0)* before to create a list of nodes.
Result is the error code. kOMS_nodeListError indicates that the list is invalid (or not initialized).
Don't try to plug when *DxOMSWorldChanged()* detected a modification in the studio... the nodes will be invalid and you could create an OMS error.

NOTE : node parameter starts at 1 :
If *DxOMSGetNodes* found 3 nodes for example in the result code, the first node will be 1, the last: 3. You can then call this code :

```
numberNodes = DxOMSGetNodes(0)
if numberNodes > 0 then
      for i = 1 to numberNodes
            result = DxOMSPlugOutNode (i)
            result = DxNoteON (1, 64, 64)
            // time passes
            result = DxNoteON (1, 64, 0)
      next
end if
```

## **result** = **DxOMSPlugInNode** ( **node** )

result as Integer
node as Integer

This call creates a connection between your DxMidi control MIDI OMS IN socket and any nodes detected with *DxOMSGetNodes(1).*
This will create a real connection.
If you call this method with different nodes, you will connect all theses nodes at the same time.
Call *DxOMSDisconnectInNode* (node) to cut the connection made.
Don't connect to a node invalid or obsolete. If you plan to memorize the nodes of your studio into file and create a prefs file for studio, verify the studio compatibility when you restart the app and prevent the user for any moves or modifications.

NOTE : nodes parameter starts at 1, like *OMSPlugOutNode().*

## **name** = **DxOMSGetNodeOutName (** node )
## **name** = **DxOMSGetNodeInName (** node )

name as String
node as Integer

This call the node listed and identified by his number after *DxOMSGetNodes(),* and reply the name. Useful to create a popup with all the nodes detected and let the user choose the nodes.
Remember that there is two lists, one for In and one for OUT.

node : the node number into the INput or OUTput list.

**name = DxOMSDisconnectINNode ( node )**
name as String
node as Integer

This call the node listed and identified by his number after *DxOMSGetNodes(),* and disconnect the INput connection between OMS and DxMidi. Remember that there is two lists, one for IN (the concerning list) and one for OUT.

node : the node number into the INputs list.

**result = DxOMSNodesINDial ( prompt , multi )**
prompt as string
multi as Boolean
result as Integer

This is a specific call made to the OMS facilities. It's not a DxMidi dialog but an OMS service. We added the capability of connecting the ports selected to the OMS IN DxMidi socket.

Call *DxOMSFilterIN()* to specify the filters.
Call *DxOMSDisconnectInNode()* to close the ports anytime.

If multi = true then you let the user selecting multiple nodes.
prompt sets the info text in the OMS dialog.
NOTE : There is no link between this call and the INputs list created with *DxOMSGetNodes(1).*

**result = DxOMSGetNodeInUniqueID ( node )**
**result = DxOMSGetNodeOutUniqueID ( node )**
result as Integer
node as Integer

You can use this to get the unique ID of a member of the nodes list (after a call to *DxOMSGetNodes()*), and keep this ID into memory or prefs to create automatic patches in your studio on starting. See examples tips and tricks to implement this. The UniqueID is a value that is not modified by the OMS dynamically. But if you change your location studio, you will need to remap the nodes again because de ID may be different. Result code is 0 when the uniqueID cannot be found.

**result = DxOMSPlugInUniqueID ( uniqueID )**
**result = DxOMSPlugOutUniqueID ( uniqueID )**
result as Integer
uniqueID as Integer

Connection is made by this function with the uniqueID obtained by *DxOMSGetNodeInUniqueID()* or *DxOMSGetNodeOutUniqueID().* result is 0 if the uniqueID cannot be found.

**result = DxOMSWorldChanged ()**
result as Boolean

If result = true then OMS prevents your application that the studio structure has changed.
You must reconsider to send *DxOMSGetNodes(0)* and *DxOMSGetNodes(1)* again.

# 7/ DxMidi - Sysex suppor t

DxMidi provides Sysex support for librarian applications that you can write. These methods let the DxMidi plug-in drives the sysex you expects and don't care of any other data. You can also decide of the minimum size of the sysex (to prevent from short initiating messages) and specify a manufacturer. With this last function, you can take care of only a specific kind of sysex.

**err** = **DxStartSysEx** ( **manufacturer** , **minsize** )
**err** = *control.***StartSysEx** ( **manufacturer** , **minsize** )
err as Boolean

First, use this method to start the Sysex mode (while you are in this mode, the driver don't receive other datas than Sysex..., you can also receive Sysex in the other mode but lang datas are not managed with the same easiness.
Set the manufacturer to filter other sysex messages. Minsize prevents smaller sysex to be filtered.

**result** = **DxIsSysEx** ()
**result** = *control.***IsSysEx** ()
result as Boolean

Call this in a timer control to test if the user has sent a sysex (or after a request).
If result = true, then call the ReadSysEx() method to get it.

**sysex** = **DxReadSysEx** ()
**sysex** = *control.***ReadSysEx** ()
sysex as String

If the sysex is here, (after your call to *IsSysEx()),* call this to get the string of the new sysex.

**err** = **DxStopSysEx** ()
**err** = *control.***StopSysEx** ()
err as Boolean

This stops the SysEx filtering function and returns to standard mode (not filtered). You must call this after the getting events procedure to let your apps take control of other datas.

# 8/ DxMidi *(V2 Pro only)* - Sequencer

The sequencer included into DxMidi is not produced directly in Realbasic, but the Real-time engine is encapsulated into the plugin. Because of this, it was more easy for us tu create a fast and easy to use engine, just driven by Realbasic methods.

**Plugin constraints**
The plugin interface is limited in memory space to include tracks. Because of this, you are limited to 16 tracks that share the entire sequencer memory.
It is not possible at this point, to get more than 1Mo of Sequencer memory tu use the sequencer. But each event is recorded in buffers with only 8 bytes. With this, it is possible to get a sequencer size like this :
(1Mo / (16 tracks + 1 mastertrack) ) / 8 bytes = 7350 events by track.

**Sequencer structur e**
DxMidi sequencer is using a specific recording engine that is used asynchronously to the Realbasic program. This let you play music and keep time to perform many other things.
The sequencer is only compatible with a PPC computer, with more than 9OMHZ.
The sequencer can play all 16 tracks simultaneously.
The time granularity is based on 192 ticks by beat.
There is no measure system, and a beat is representing a musical tempo event.
A tempo of 60 represents 192x60 ticks by minute.
This granularity is enough precise to play song without problems.

**Time expressions**
Further methods use 2 params to define the time locator :`
A beat value (integer)
A fraction (tick) value (integer).
When the fraction part is 192, then it is equivalent to :
beat+1 , 0
Some other methods need to reply with a time information/
Only in this case, the return value will be expressed in «fracs» values (just count 192 ticks for a beat, and do the conversion).

**Input Output synchronisation**
For now (V2.0ß1), there is no external or internal MIDI timecode. The sequencer is only driven by an internal timer system.
The MIDI time code will be used in a next release of DxMidi.

**Sequencer - opening and closing**
Before starting the sequencer, you must have a valid OMS connection. If not, the sequencer cannot process its engine and record/play notes.
First, use the *DxOpenSequencer()* method when you are sure to have a valid OMS connection.
Then you can use all DxSequencer methods...
At the end, you must close the sequencer by using the *DxCloseSequencer()* method.

The sequencer use these routines to create a memory space for the sequencer, and at the end, to release the memory to the system.

**Quit condition**
In specific conditions (sequencer is active, and playing), any error that generates a crash, could crash the sequencer too... and maybe the OMS driver... in most of these cases, restart the computer.

Before creating a big application with sequencer, try to test the main fonctions with :
- OMS connection
- Open Sequencer
- Set the locator
- Play the tracks
- Stop the sequencer
- Close sequencer
- release OMS connection

If all of these commands are correctly handled in window events, you will have no more trouble with fatal errors and crashes.

If you plan to use a window application :
For *DxOpenSequencer,* the best place is an «Open» window event
For *DxCloseSequencer,* the best place is a «Close» window event
If you are using an Application class, put together OMS & Sequencer opens in «Open» event, and OMS & Sequencer closes in the «Close» event.

The quit command is abstractive with using DxMidi handlings, it would be better to use it once in your application.

**Troubleshooting**
If you have problems with MIDI events between Sequencer and your MIDI configuration, first try to test your midi studio with standard DxMidi methods : DxNoteON, DxSendControl...

**Mastertrack**
This tool let you specify tempi changing like a check list.
beat+1 , 0

**Input Output synchronisation**
For now, there is no external or internal MIDI timecode. The sequencer is only driven by an internal timer system.

The MIDI time code (MTC) will be used in a next release of DxMidi.

## Open/close methods

### rep = DxOpenSequencer ( mem )

mem as integer
rep as boolean

This, reserve memory for the sequencer, and initialize many settings and default values.
The mem size is expressed as KILO-BYTES.
example : to reserve 500K, type : rep = DxOpenSequencer(500)
NOTE : Never set a value under 50K...

rep replies FALSE if the sequencer cannot reserve this memory.

### rep = DxCloseSequencer ()

rep as boolean

This closes the sequencer, and release the memory reservation.
rep replies FALSE if DxMidi detected an unknown error.

## Locator

static Boolean DxMidiSeqSetTime(int beat, int time)

### rep = DxMidiSeqSetTime ( beat , frac )

beat as integer
frac as integer
rep as boolean

Before starting to play the sequencer, you must initialize the time setting. The best is to initialize with Beat 0 and Frac 0.
rep = DxMidiSeqSetTime(0,0)
rep is a dummy reply.
beat and frac represents the time position.

time = **DxMidiSeqGetTime** ()
time as integer

Use this call to get the current locator position.
Time is a frac value of the complete time locator.
To parse Beat & Frac value, type this code :

```
time = DxMidiSeqGetTime()
myBeat = time \ 192
myFrac = time mod 192
```

NOTE : \ is an integer division.

## Transport methods

rep= **DxMidiSeqPlay** ()
rep as boolean

This call trigger the sequencer play at the locator position.
rep replies an error (false) if the sequencer is not ready to play or not activated.

rep= **DxMidiSeqStop** ()
rep as boolean

This call stops the sequencer play. The locator is keeping the stop time location until the sequencer is restarted, or moved by any other function *(DxMidiSeqSetTime()* for example).
rep replies an error (false) if the sequencer is not ready to play or not activated.

## Recording

To record a track in DxMidi sequencer, you must select first a recording track.
By default, there is no track selected. You must choose this track by sending the *DxMidiSeqRecTrack()* method.
By this, the last time you play the sequencer, the selected track will be in recording mode.
To stop the recording mode for the track, use : *DxMidiSeqKillRec().*

As you can see, the is no «recording» method... but just a «mode» that let you record on a track like a punch in/punch out..
If you are playing the sequencer, and send a *DxMidiSeqRecTrack()* method just in time, the track selected will be recording datas.
If always in just in time mode, you send *DxMidiSeqKillRec(),* the track

is stopping the recording, and the track returns in playing mode.
Like this.

Note : in recording mode, the recorded track is not playing.

### rep = **DxMidiSeqRecTrack** ( track )

rep as integer
track as integer

With this, you are setting the track for recording. If the sequener is playing, you are immediately in recording mode on this track.
rep is not use at this time. Future use only.
The track is a number between 1 and 16.

### rep = **DxMidiSeqKillRec** ()

rep as integer

Stops any recording on a track. rep will be used in a future version.

### **rep** = **DxMidiSeqNote** ( **track** , **beat** , **frac** , **note** , **dur** , **vel** )

track as integer
beat as integer
frac as integer
note as integer
dur as integer
vel as integer
rep as integer

This is a different recording system.
NEVER use this while playing the sequencer.
This method insert a note in the «Time» place in the track specified.
If you want to insert a complete note, call just this method once.
In fact, you indicates a time lenght to your note, and DxMidi insert two events : the note ON and the note OFF in place.
track is the target track to insert the note.
beat and frac is the time position you must set you are free to specify any value, but try to respect the min and max values of frac (0-192).
note is the MIDI note value (0-127).
dur is the note lenght expressed in frac complete value.
To set this frac value, use this calculation (example) :

```
mybeat = 54
myfrac = 24
mycompletefrac = (myBeat * 192) + myfrac
```

You are just limited in length by the integer limitation itself.
vel is the note velocity. rep relies an error if the track is memory full.

## Track specific methods

Other methods are used to manage tracks and their playing mode.

### rep= **DxSeqSetTrackChannel** ( track , channel )

track as integer
channel as integer
rep as boolean

The track channel is set with this method.
When DxMidi sequencer is initialized, all tracks are set to channel 1.
track is the target track (1-16)
channel is the channel number (1-16)
rep replies false if a value is out of range.

### rep= **DxMidiSeqClearTrack** ( track )

track as integer
rep as boolean

This performs a complete erase of the track content.
track is the target track (1-16)
rep replies false if the track value is out of range.

### rep= **DxMidiSeqMuteTrack** ( track , mutemode )

track as integer
mutemode as integer
rep as boolean

This performs a mute / unmute on the track.
This don't modify the track itself, but only the playing engine mode.
if mutemode is TRUE : track is muted.
if mutemode is FALSE : track is unmuted.
track is the target track (1-16)
rep replies false if the track value is out of range.

## Tempo and metronome

You will find here methods to control the tempo of the sequencer, and how to play a metronome while playing/recording.

### rep= **DxMidiSeqTempo** ( **tempo** )

tempo as integer
rep as boolean

Set the tempo of the sequencer, even during playing or recording.
rep replies false if the tempo value is out of range.

### rep= **DxMidiSeqMMSetting** ( **chan** , **note** , **vel** )

chan as integer
note as integer
vel as integer
rep as boolean

The metronome is always playing in Midi on a specified (chan) channel, note and with a specific (vel) velocity.
rep replies false if the tempo value is out of range.

### rep= **DxMidiSeqMMAction** ( **action** )

action as boolean
rep as boolean

Set the metronome activity. if action is true, the metronome is playing on the MIDI channel, note and velocity defined by the *DxMidiSeqMMSetting()* method.
rep is unused.

## Mastertrack

The DxMidi sequencer mastertrack is simply a 17th track that records tempo changes.
If you want use the mastertrack during playing, set the *DxMidiSeqMTAction()* method to true. If not, just the global tempo defined for DxMidi Sequencer will be used.
The mastertrack is a list with tempo events.
To insert a tempo event, use *DxMidiMTInsert().* To clear a tempo event, use *DxMidiMTDelEvent().* To To get each element of your mastertrack list, first get the number of events in the list using *DxMidiMTGetSize(),* and then, get each event with *DxMidiMTGetInfoTime()* and *DxMidiMTGetInfoTempo().*
Use *DxMidiMTClear()* to clear the entire content of the mastertrack.
Use *DxMidiSetInfoTempo()* to redefine an existing tempo in an event of the list.

### rep= **DxMidiSeqMTAction** ( **action** )

action as boolean
rep as boolean

Set the mastertrack activity.
if action is true, the mastertrack is activated.
rep is unused.

### rep= **DxMidiMTClear** ()

rep as boolean

Erase the all content of the mastertrack. This can't be undone.
rep is unused.

### rep= **DxMidiMTInser t** ( beat , frac , tempo )

beat as integer
frac as integer
tempo as integer
rep as boolean

This call inserts a new event in the list. The location time is specified with a beat and frac value. This can be freely defined.
tempo contains the tempo value for this event.
rep reply False if the track memory is full.

### size = **DxMidiMTGetSize** ( )

size as integer

size reply the number of events in the mastertrack list.
This method is used when you want to make a listbox with events of the mastertrack.

### time = **DxMidiMTGetInfoTime** ( pos )

time as integer

This method replies the time position of the specified event.
Pos is the position number of the event in the list.
The time value is expressed in fracs. To get the beat value, use the calculation :

```
mybeat = time \ 192
myfrac = time mod 192
```

### time = **DxMidiMTGetInfoTempo** ( pos )

time as integer

This method replies the tempo value of the specified event.
Pos is the position number of the event in the list.

### rep = **DxMidiMTSetInfoTempo** ( pos , tempo )

pos as integer
tempo as integer
rep as boolean

If you want to modify an existing tempo event, use this call.
tempo is the tempo value.
Pos is the position number of the event in the list.
rep replies false is pos is out of range.

### rep = **DxMidiMTDelEvent** ( pos )

pos as integer
rep as boolean

Use this to delete a specific event in the mastertrack.
Pos is the position number of the event in the list.
rep replies false is pos is out of range.

## Track services

These methods are useful to complete your sequencer. It contains methods to get the track content as a string. To set the track content with a string. To get the track event size, and to paint a visible event «view» of your track.

### content = **DxMidiGetTrack** ( track )

content as string
track as integer

Use this to get the track entire content as a string.
This is useful if you want to save your track into a file.
Don't forget to keep the track event size too and save it into your file.
content is the track content as string.
track is the track number.

### size = **DxMidiGetTrackSize** ( track )

size as integer
track as integer

Use this method to get the number of events in the track.
DxMidi stores an event into 8 bytes.
size replies -1 if the track number is out of range.

### rep = **DxMidiSetTrack** ( track , content , size )

track as integer
content as string
size as integer
rep as boolean

Use this to replace the entire content of memory track with a string content.
This is useful if you want to load your track from a file and then store it in your track.
Don't forget to get the track event size from your file, and set it as the size parameter.
content is the track content as string.
track is the track number.
rep replies an error if the track memory is full.

rep = **DxMidiPaintTrack** ( **track** , **X** , **Y** , **L** , **H** , **time** , **pixsize** )
track as integer
x as integer
y as integer
l as integer
h as integer
time as integer
pixsize as integer
rep as boolean
This routine draws a portion of your track as a graphic representation if its content. You can define the rect of your clipping preview, and force the routine to only draw events from any time location, with a scale ratio, and to the right of the rectangle.

track : specify the track content to read and preview.
x,y : position of the left/top of the rectangle to preview, as a global window pixel value (from the left/top corner of the current parent window).
l,h : width and height of your preview rectangle.
time : the location time from where the preview is starting (specified in a complete frac time expression).
pixsize : is a zoom level from 1 (smallest) to any value (zooming value). A medium pixsize is near 30 as pixsize.
rep is unused.

# 9/ DxMidi -Result Codes

**Error codes :**

```
#define kDxMM_Unavailable -1
#define kDxOMS_Unavailable -101
```
The driver (MM or OMS) is unavailable, not installed or the serial port is used yet.

```
#define kMM_SignatureError -2
#define kOMS_SignatureError -102
```
Signature error on MidiConnect command (MM or OMS)

```
#define kMM_Absent -10
#define kOMS_Absent -110
```
Signature error on MidiDisconnect command (MM or OMS)

```
#define kMM_CloseError -11
#define kOMS_CloseError -111
```
Signature error on closing ports

```
#define kMM_MemError -33
#define kOMS_MemError -133
```
Memory error on creating events buffers.

**Alert codes :**

```
#define kMM_OutAlert 2
```
Midi OUT connection aborted(no connection made)

```
#define kMM_InAlert 4
```
Midi IN automatic connection aborted (no connection made)

```
#define kMM_PatchAlert 8
```
Midi Patch alert - le patchBay (MM) detected an internal error

```
#define kMM_NoErr 0
#define kOMS_NoErr 0
```
Midi no error

```
#define kMM_Present 100
#define kOMS_Present 200
```
Midi double SignIn alert
(you just have to apply Connect() once on the control)

## OMS sendData constants

```
#define omsContMask 0x03
#define omsNoCont 0x00
#define omsStartCont 0x01
#define omsMidCont 0x03
#define omsEndCont 0x02
```

## Midi nodes types (extracted from OMS (C SDK)

```
enum {
omsIncludeInputs = 1,
omsIncludeOutputs = 2,
omsIncludeReal = 4,
omsIncludeVirtual = 8,
omsIncludeSync = 16,
omsIncludeSyncOnly = 32,
omsMergeOutputsByPort = 64,
omsIncludeSecret = 128
};
```

## MidiManager constants for SendData

```
midiNoCont = 0x00,
midiStartCont = 0x01,
midiMidCont = 0x03,
midiEndCont = 0x02,
```

Report bugs : realvision@cabanis.com
Tech info : realvision@cabanis.com

DxMidi plugin is shareware. If you are using this for your programs, pay your fee
Standard edition (20$) contribution.
Pro edition (35$) contribution.
If you use this plugin for commercial use, don’t forget to stamp your program with the «Made with DxMidi».
The worldwide license is free for this program. You can distribute, and sell programs incluing DxMidi V2 Pro freely.

RealVision WebSite :
**http://www .cabanis.com/r ealvision/**