MIDI-OX REXX Interface

You can run REXX and Object-REXX programs from MIDI-OX. The programs can access MIDI data provided by a named REXX queue. The data can be processed in any fashion and then sent out the MIDI output port via MIDI-OX supplied functions.

MIDI-OX does not supply REXX, but will use the environment if available.

The Queue

MIDI-OX uses a standard REXX queue construct. In fact, it is REXX itself that provides the queue. Data is entered into the queue by MIDI-OX in a FIFO (first-in, first-out) order, but the REXX program can remove the data in anyway is wants.

MIDI-OX will suggest a queue name that will be unique within the MIDI-OX instance. It is possible to override this name, even with the name of an existing queue in another MIDI-OX instance. Since entries are de-queued by the REXX program when they are read, only one instance will read any particular message.

Real-time MIDI messages are entered into the queue as they are received, as long as the REXX program is running. By default, the data entered is the standard MIDI message: a character representation of the decimal data bytes. The representation can be changed by a function call **MOXSetQueueDataFormat()**, which is described later. Each queue entry message is three or four data values, possibly padded with zeros for MIDI messages that are shorter than three bytes. No running status is supplied or should be expected. A millisecond timestamp is also provided. This represents the time elapsed since the MIDI device was opened.

Examples:
15632 144 64 106 Note On at 15632 millisecs: channel 1, note #64 (E4), velocity 106
2338 192 33 00 Program Change at 2338 millisecs: channel 1, patch #33

The data can easily be parsed into separate variables and processed. For example:

```
PULL Status Data1 Data2
```

The above will move the next queue entry into the four variables named.

The standard REXX conversion functions can be used to convert between data formats. For instance you could split off the event and channel information from the status by:

```
Event = BitAnd( X2C( D2X( Status ) ), 'F0'x )
Chan = BitAnd( X2C( D2X( Status ) ), '0F'x )
```

If you always want the event and channel split off, you can have MIDI-OX do it for you by an alternate queue data format:

```
qName = MOXSetQueueDataFormat( "C" )
...
PULL TimeStamp Event Chan Data1 Data2
```

The above will split off the event and channel information, and return the data in decimal format.

When MIDI-OX is requested to end a REXX program it will queue a special value, that alerts the REXX program to exit ('**END_DATA**'). If the program does not respond in a reasonable amount of time, you can forcibly terminate it. This prevents a stuck program from languishing.

MIDI Output

Once the data is processed, it can be returned to MIDI-OX for output. You use the **MOXOutputMIDI()** function to return the data.

Example:
ret = MOXOutputMidi(Status, Data1, Data2)

Alternate functions exist to return the data channelized and/or in Hex:

ret = MOXOutputMidiC(Event, Chan, Data1, Data2)

Expects a MIDI status in **Event** and the channel information in **Chan**. MIDI-OX combines it to form the output message.

MIDI-OX REXX Programs

The normal way a MIDI-OX REXX program operates is by polling the queue for data. If any is available, it is processed and output; if not, it should do any background tasks and then block while waiting for more input. A system semaphore is created with the same name as the queue and is pulsed each time MIDI data is queued.

Example:

```
/* REXX: Simple MIDI output loop */
call RxFuncAdd 'SysLoadFuncs', 'RexxUtil', 'SysLoadFuncs'
call SysLoadFuncs

qName = MOXGetQueueName()
oldq  = RxQueue( 'Set', qName )

/* MIDI-OX creates a semaphore with same name as queue */
sem = SysOpenEventSem( qName )
if sem = 0 then
   running = 0
else
   running = 1

do while running
   do while Queued() <> 0
      pull timestamp status data1 data2
      if timestamp = 'END_DATA'
      then do
         running = 0
         leave
      end

      /* ... do any processing.  Then ... */
      ret = MOXOutputMIDI( status, data1, data2 )
      if ret <> 0
      then do
```

```
            call RxMessageBox( MOXErrorText( ret ), , , 'STOP' )
         end
      end

      if running
      then do
         /* queue is empty: you could do other processing here */
         ret = SysWaitEventSem( sem )
         if ret <> 0 then
            running = 0
      end
   end

   call SysDropFuncs
   exit
```

If a parsing or runtime error occurs in the REXX program, the error or trace diagnostic is displayed in the **Actions | Run REXX...** dialog box. This information is invaluable for debugging REXX programs. Several REXX programs are installed during installation: these examples should be useful as a starting point for your own programs.

MIDI-OX REXX Functions

MIDI-OX REXX API functions are registered with REXX before a program begins, so they are always loaded and available to run in an efficient manner. The following routines are defined:

MOXGetQueueName( )

Summary: Returns the name of the private data queue that MIDI-OX creates for REXX programs. This name is used for both the data queue, and for the name of a system semaphore that is used to wake up the REXX program when data is available.

MOXSetQueueDataFormat(format)

Summary: sets the format for the data entered into the queue. Calling this function will clear the queue of any existing data.

format: one of the following:

- 'N' (Default) Queue values in decimal. Example **147 64 120**
- 'C' split off the channel information from the event type. This causes an extra channel field to be inserted into the message. Example: **event chan data1 data2 – 144 3 64 55**
- 'X' output all values in Hexadecimal. Example: **status data1 data2 – 93 40 6F**
- 'S' Like 'C', but data is output in Hex: **90 03 40 4F**

Return: This function returns the private data queue name if sucessful.

Notes: This function is optional. It has the side effect of emptying (discarding) the queue each time it is called.

MOXGetSystemTime( )

Summary: returns the result of the standard windows MME API, timeGetTime(), as a numeric text string. The value represents the number of milliseconds elapsed since the system was started.

MOXErrorText(err)

Summary: Returns an error message text string associated with the error number supplied. If an unknown error number is input 'Unknown Error - #**err**' results.

Err: a standard MIDI-OX REXX error number.

Number	Text	Description
100	Bad parameter: number required	Non-numeric input passed to routine.
101	Bad parameter: out of range	Value was out of legal range
102	Bad parameter: input required	Missing data
103	Bad parameter: unknown status	Illegal MIDI status
104	Bad Parameter: unexpected data	Parameter not allowed
105	Bad Parameter: unknown Argument	Unrecognized parameter
106	Error: out of memory	System Memory Error

MOXOutputMidi(Status, Data1, Data2)

Summary: Sends a MIDI message to MIDI-OX for output to the currently open port. All data is assumed to be in decimal format: an error will result if it's not.

Status: A combined event-channel message number. The standard MIDI spec. event type (Most significant 4 bits) and the zero based channel number (least significant 4 bits).

Data1: Depends on message type, but always a value between 0 and 127 inclusive. For a Note event the first data value is the MIDI note number; for a Program change it is the patch number.

Data2: Depends on message type, but always a value between 0 and 127 inclusive. For a Note event the second data value is the MIDI note number; for a Program change it is not used (and should be supplied as 0).

Return: This function returns 0 for success, or an error code for failure.

MOXOutputMidiC(Event, Chan, Data1, Data2)

Summary: Sends a MIDI message to MIDI-OX for output to the currently open port. All data is assumed to be in decimal format: an error will result if it's not.

Event: The standard MIDI spec. event type. Only the most significant 4 bits of this number will be set – the least significant bits will be 0.

Chan: The zero based channel number. This is a number between 0 and 15, that represents MIDI channels 1 through 16.

Data1: Depends on message type, but always a value between 0 and 127 inclusive. For a Note event the first data value is the MIDI note number; for a Program change it is the patch number.

Data2: Depends on message type, but always a value between 0 and 127 inclusive. For a Note event the second data value is the MIDI note number; for a Program change it is not used (and should be supplied as 0).

Return: This function returns 0 for success, or an error code for failure.

MOXOutputMidiX(Status, Data1, Data2)

Summary: Sends a MIDI message to MIDI-OX for output to the currently open port. All data is assumed to be in Hexadecimal format: an error will result if it's not.

Status: A combined event-channel message number. The standard MIDI spec. event type (Most significant 4 bits) and the zero based channel number (least significant 4 bits).

Data1: Depends on message type, but always a value between 00 and 7F inclusive. For a Note event the first data value is the MIDI note number; for a Program change it is the patch number.

Data2: Depends on message type, but always a value between 00 and 7F inclusive. For a Note event the second data value is the MIDI note number; for a Program change it is not used (and should be supplied as 0).

Return: This function returns 0 for success, or an error code for failure.

MOXOutputMidiS(Event, Chan, Data1, Data2)

Summary: Sends a MIDI message to MIDI-OX for output to the currently open port. All data is assumed to be in Hexadecimal format: an error will result if it's not.

Event: The standard MIDI spec. event type. Only the most significant 4 bits of this number will be set – the least significant bits will be 0.

Chan: The zero based channel number. This is a number between 00 and 0F, that represents MIDI channels 1 through 16.

Data1: Depends on message type, but always a value between 00 and 7F inclusive. For a Note event the first data value is the MIDI note number; for a Program change it is the patch number.

Data2: Depends on message type, but always a value between 00 and 7F inclusive. For a Note event the second data value is the MIDI note number; for a Program change it is not used (and should be supplied as 00).

Return: This function returns 0 for success, or an error code for failure.

MOXOutputSysEx(Data)

Summary: Sends a MIDI System exclusive message to MIDI-OX for output to the currently open port. All data is assumed to be in Hexadecimal format: Hex bytes can be separated by spaces. Example: **F0 41 10 42 12 40 00 7F 00 41 F7**

Return: This function returns 0 for success, or an error code for failure.

Notes: The function does not actually test the data for validity other than to send only valid Hex data. This means that it is the programmer's responsibility to ensure that SysEx messages are prefixed by 'F0' and suffixed by 'F7'. It also means that you can send any MIDI data with this function call, but it will be sent as a Windows long MIDI message.