Help for MIDI I/O

MIDI Input Properties	MIDI Input Events
MIDI Output Properties	MIDI Output Events

Registration Information

Order Form

Getting Custom Controls Written

Licensing Information

MIDI Input Description

The MIDIIN control is used to receive MIDI messages from external MIDI devices. Messages can be retreived using the Message event or polling.

In the MIDIThru sample project, we use both methods (events and polling) together. When a Message event is fired, we process that event and then poll the MIDIIn control for other waiting messages. In this example, stud the Do While loop in the Message event.

All messages are immediately time-stamped as the control receives the event with millisecond accuracy. The time stamp is automatic and takes place in the control, not in Visual Basic. This results in a highly accurate time stamp. The MIDIIN control has an internal queuing mechanism so if messages arrive faster than your application can handle them they will not be lost.

The MIDI IN Custom Control automatically detects the available MIDI IN ports and allows you to select the port you wish to use. As MIDI events are received through the selected MIDI IN port, each MIDI event receives a high resolution time stamp. A built in message queue allows Long and Short MIDI messages to be received.

- Receive MIDI messages from external MIDI devices
- Messages can be retrieved using events or polling
- Messages are time-stamped with millisecond accuracy
- Built in internal queuing mechanism so if messages arrive faster than your application can handle them, the messages will not be lost
- Buffer for receiving system exclusive messages
- Automatically detects the available MIDI IN ports

MIDI Output Description

The MIDIOut control gives you complete control over the contents and timing of MIDI messages sent to either internal or external MIDI devices. All timing is in milliseconds.

You can queue as many messages as you like (within the constraints of available memory) before starting output, or you can queue one or more messages prior to starting output and then add more as the output proceeds. Messages can be placed into the MIDI Output queue in any timing order. The MIDIOut control automatically sorts the messages into the correct order.

Messages are scheduled for transmission at a time you specify relative to the time that output is started. As with the MIDIIn control timing has millisecond resolution, giving you the ability to precisely control the timing of sent MIDI messages.

The MIDI OUT Custom Control automatically detects the available MIDI OUT ports and allows you to select a MIDI OUT port to use. This control allows MIDI messages to be sent directly to the MIDI OUT port and also allows MIDI messages to be queued for playback at the appropriate time.

- Complete control over the contents and timing of MIDI messages sent to either internal or external MIDI devices.
- Queue as many messages as you like (within the constraints of available memory) before starting output.
- Queue messages prior to starting output and then add more as the output proceeds.
- Messages are scheduled for transmission at a time you specify relative to the time that output is

started.

- Control timing has millisecond resolution, giving you the ability to precisely control the timing of sent MIDI messages.
- Automatically detects the available MIDI OUT ports.
- Complete support for system exclusive messages.
- Timer event provides a high-resolution timer.
- Automatically detect whether or not the current device supports patch caching.
- Automatically detect whether or not the current device supports volume and allows for control of devices master left/right volume.
- Adjust tempo playback rate in realtime.

VBX Compatibility

VB 2.0 and up

File Name

MIDIIO.VBX, MIDIIO16.OCX, MIDIIO32.OCX

Object Type

MIDIInput

MIDIOutput

Remarks

We strongly advise that you purchase a copy of one of more of the MIDI specifications. To purchase these specifications, you should contact the MIDI Manufacturers Association.

MIDI Manufacturers Association
Post Office Box 3173
La Habra, CA 90632

Phone: 310-947-8689
Fax: 310-947-4569

You can also purchase MIDI books from the Mix Bookshelf. They sell a wide variety of books about MIDI and music.

Mix Bookshelf
6400 Hollis Street
Suite 12
Emeryville, CA 94608

Phone: 800-233-9604 or 510-653-3307
Fax: 510-653-5142

Microsoft sells three books that are specifically aimed at multimedia development on Windows. We have found these books to be quite valuable. The books are:

Microsoft Windows Multimedia Programmer's Reference
Microsoft Windows Multimedia Programmer's Workbook
Microsoft Windows Multimedia Authoring and Tools Guide

United States: 800-MSPRESS
Canada: 416-293-8141
Other Locations: Contact your local Microsoft office

Distribution Note When you develop and distribute an application that uses this control, you should install the control into the users Windows SYSTEM directory. This control has version information built into it. So, during installation, you should ensure that you are not overwriting a newer version.

Registration Information

Credits

The MIDI Pack was written by James Shields and Zane Thomas.

Inquiries, tech support, comments should be sent to Mabry Software. Our address is 71231,2066 on CompuServe, or mabry@mabry.com on Internet. You can call us at 2066341443 or fax us at 206632-0272. If you need to send something via U.S. Mail, the address is:

Mabry Software, Inc.
Post Office Box 31926
Seattle, WA 98103-1926

Registration

You can register this program by sending $99 ($104 for international orders) and your address. You can register the MIDI Pack **and** its C++ source code by sending $299 ($304 for international orders). With your order, you will receive a copy of our manual documenting all of the MIDI Pack controls.

Add $5 per order for shipping and handling.

For your convenience, an order form has been provided that you can print out directly from this help file.

Prices are subject to change without notice.

E-mail Discount

You may take a $5 discount for e-mail delivery of this package (CompuServe or Internet). If you choose this option, please note: a printed manual is not included. Be sure to include your full mailing address with your order. Sometimes (on the Internet) the package cannot be e-mailed. So, we are forced to send it through the normal mails.

CompuServe members may also take the $5 e-mail discount by registering this package in the software registration forum (GO SWREG). The MIDI Packs SWREG ID number is 9525. The source code version's ID number is 9528.

Credit Card Orders

We accept VISA, Mastercard and American Express. If you e-mail your order to us, please be sure to include your card number, expiration date, complete mailing address, and your phone number (in case we have any questions about your order).

© Copyright 1995-1997 by Mabry Software, Inc.

MIDI I/O Order Form

Use the Print Topic.. command from the File menu to print this order form.

Mail this form to:	Mabry Software, Inc. Post Office Box 31926 Seattle, WA 98103-1926 Phone: 206-634-1443 Fax: 206-632-0272 CompuServe: 71231,2066 Internet: mabry@mabry.com

Where did you get this copy of MIDI I/O?

____

Ship to: ____

____

____

____

____

Phone: ____

Fax: ____

E-Mail: ____

MC/VISA/AMEX: __________________________________ exp. _______________

P.O. # (if any): ____

qty ordered	____	REGISTRATION $40 each ($45 international). Check or money order in U.S. currency drawn on a U.S. bank. Add $5.00 per order for shipping and handling.
qty ordered	____	SOURCE CODE AND REGISTRATION $120 each ($125 international). Check or money order in U.S. currency drawn on a U.S. bank. Add $5.00 per order for shipping and handling.

MIDI Input Properties

All of the properties that apply to this control are in this table.   Properties that have special meaning for this control or that only apply to this control are marked with an asterisk (*).

***Action** Property
Align Property
***Buffer** Property
***Data1** Property
***Data2** Property
***DeviceCount** Property
***DeviceID** Property
***DriverVersion** Property
Enabled Property
***HMidiDevice** Property
Index Property
Left Property
***ManufacturerID** Property
***MaxSysexSize** Property
***MessageCount** Property
***MessageEventEnable** Property
***Message** Property
Name Property
***ProductID** Property
***ProductName** Property
***State** Property
Tag Property
***Time** Property
Top Property

MIDI Input Events

All of the events that apply to this control are in this table.   Events that have special meaning for this control or that only apply to this control are marked with an asterisk (*).

***Error** Event

***Message** Event

MIDI Output Properties

All of the properties that apply to this control are in this table.   Properties that have special meaning for this control or that only apply to this control are marked with an asterisk (*).

***Action** Property
Align Property
***Buffer** Property
***CanCache** Property
***Data1** Property
***Data2** Property
***DeviceCount** Property
***DeviceID** Property
***DeviceType** Property
***DriverVersion** Property
Enabled Property
***HasLRVolume** Property
***HasVolume** Property
***HMidiDevice** Property
Index Property
Left Property
***ManufacturerID** Property
***Message** Property
***MessageTag** Property
Name Property
***Notes** Property
***PlaybackRate** Property
***ProductID** Property
***ProductName** Property
***State** Property
Tag Property
***Time** Property
Top Property
***Voices** Property
***VolumeLeft** Property
***VolumeRight** Property

MIDI Output Events

All of the events that apply to this control are in this table.   Events that have special meaning for this control or that only apply to this control are marked with an asterisk (*).

***Error** Event

***MessageSent** Event

***QueueEmpty** Event

***Timer** Event

Action Property (MIDI Input Control)

See Also Example

Description

Action to take using current DeviceID.

Usage

[*form.*][*control.*]**Action**[= *integer*]

Remarks

Setting this property causes an action to occur using current DeviceID. The actions are:

Constant	Value	Meaning
msMIANone	0	No action
msMIAOpen	1	Open device
msMIAClose	2	Close device
msMIAReset	3	Reset MIDI device.
msMIAStart	4	Start MIDI input
msMIAStop	5	Stop MIDI input
msMIARemove	6	Remove current MIDI message from queue

Data Type

Integer

See Also

Properties:

Action (MIDI Output)

Action Property (MIDI Output Control)

See Also Example

Description

Action to take using current DeviceID.

Usage

[*form.*][*control.*]**Action**[= *integer*]

Remarks

Setting this property causes an action to occur using current DeviceID. The actions are:

Constant	Value	Meaning
msMOANone	0	No action
msMOAOpen	1	Open device
msMOAClose	2	Close device
msMOAReset	3	Reset MIDI device.
msMOAStart	4	Start MIDI output
msMOAStop	5	Stop MIDI output
msMOAQueue	6	Queue message given by Message, Data1, and Data2 will be queued for playing at Time milliseconds after output is started (Action = 4)
msMOAImmediate	7	Immediate. Sends the message given by Message, Data1, and Data2 immediately if output is started (Action = 5)
msMOATimer	8	Timer. Fires a Timer event when Time milliseconds have elapsed. This provides a high-resolution timer for you to use. When the Timer event is fired, it will pass back to you the contents of the MessageTag property in effect at the time that action was set to Time.
msMOAPause	9	Pauses the sending of queued message and stops the queue timer clock.

Data Type

Integer

See Also

Properties:

Action (MIDI Input)

Buffer Property

Example

Description

Holding area for system exclusive messages.

Usage

[*form.*][*control.*]**Buffer**[= *string*]

Remarks

When sending or receiving a System Exclusive (Sysex) message the buffer property is used to transfer the contents of the Sysex message.   The contents of Sysex messages is determined solely by the MIDI device sending or receiving the sysex message.

It is important to note that there is a subtle difference between the way the Buffer property is used in the MIDI File control and the MIDI In and Out controls.

When you transmit a Sysex message to a midi device using the MIDI Out control you will need to supply the sysex start and end bytes (&HF0 and &HF7) as message delimiters.

In this example, a Sysex message is sent which resets the Roland SoundCanvas SC-88 to General MIDI mode:

```
MidiOutput1.Message = &HF0

MidiOutput1.Buffer = Chr$(&HF0) + Chr$(&H7E) + Chr$(&H7F) + Chr$(9) +
     Chr$(1) + Chr$(&HF7)

MidiOutput1.Action = MIDIOUT_SEND
```

In this example the first and last bytes (&HF0 and &HF7) signal the beginning and end of a Sysex message.   The middle bytes are the Sysex messages contents.

When you receive a sysex message using the MIDI In control the start and end bytes will be the first and last bytes in the string contained by the Buffer property.

However when you read a sysex message from the MIDI File control the start and end bytes will NOT be in the string contained by Buffer.   So to transmit a sysex message retrieved from the MIDI File control you should use something like:

```
sysexMsg = &HF0 + MIDIFile1.Buffer + &HF7
```

Data Type

String

CanCache Property

Description

Specifies whether or not the current device supports patch caching.

Usage

[*form.*][*control.*]**CanCache**

Remarks

This property is read-only.

Data Type

Integer (boolean)

PlaybackRate Property

Description

Determines the speed that music plays.

Usage

[*form.*][*control.*]**PlaybackRate**[= *integer*]

Remarks

This property determines how fast music plays. Setting this property to a negative number slows the music down. Setting this property to a positive number speeds the music up. Setting this property to zero (0) makes the music play at its normal rate.

Examples: Setting this property to 100 makes the music play at double speed, 200 is triple speed, 300 is quadruple speed, etc.. -100 results in half-speed, -200 is one-third speed, -300 is one-quarter speed, etc.

You do not need to set this property to even multiples of 100, you can set it to 123, -5, or any other legitimate short integer value (-32768 to 32767).

Data Type

Integer

Clocks Property

Description

Number of MIDI clocks in a metronome click.

Usage

[*form.*][*control.*]**Clocks**[= *integer*]

Remarks

Valid only after a Time Signature meta-event (&H58) becomes the current message.   Once the values are loaded from a Time Signature meta-event they remain valid until another Time Signature meta-event is encountered.

Data Type

Integer (0 - 255)

Data1 and Data2 Properties

See Also Example

Description

MIDI message data bytes.

Usage

[*form.*][*control.*]**Data1**[= *integer*]
[*form.*][*control.*]**Data2**[= *integer*]

Remarks

The contents of Data1 and Data2 depend on the type of MIDI message being sent/received.

If the Message property is 255 (for a meta event), MsgText is loaded with a string. Data1 determines what the string represents. The following table lists the possible values:

Constant	Value	Meaning
msMFMTText	1	Non-specific text string
msMFMTCopyright	2	Copyright notice
msMFMTTrack	3	Sequence/track name
msMFMTInstrument	4	Instrument name
msMFMTLyric	5	Lyric
msMFMTMarker	6	Marker
msMFMTCuePoint	7	Cue point
	8-15	Undefined text string

Data Type

Integer (0-255)

See Also

Properties:

Message

DeviceCount Property

Example

Description

Returns the number of MIDI devices.

Usage

[*form.*][*control.*]**DeviceCount**

Remarks

This property determines the number of MIDI devices available.   Note that the number of input devices may not be the same as the number of output devices.

This property is read-only.

Data Type

Integer

DeviceID Property

See Also Example

Description

Determines the device to use.

Usage

[*form.*][*control.*]**DeviceID**[= *integer*]

Remarks

In the MIDI output control this property ranges from -1 through DeviceCount - 1, a value of -1 represents the MIDI mapper and all other values represent a MIDI device.

In the MIDI input control this property ranges from zero through DeviceCount - 1, with all values representing MIDI devices.

Data Type

Integer

See Also

Properties:

DeviceCount

DeviceType Property

See Also

Description

Type of device currently selected.

Usage

[*form.*][*control.*]**DeviceType**

Remarks

Specifies type of device selected by DeviceID.   Values are:

Constant	Value	Meaning
msMODTHardware	1	MIDI hardware port
msMODTSynth	2	General internal synthesizer
msMODTSquare	3	Square wave synthesizer
msMODTFMSynth	4	FM synthesizer
msMODTMapper	5	MIDI mapper

This property is read-only.

Data Type

Integer

See Also

Properties:

DeviceID

DriverVersion Property

See Also

Description

Driver version of DeviceID.

Usage

[*form.*][*control.*]**DriverVersion**

Remarks

This property returns the driver version number for the device specified by DeviceID.   The high-byte contains the major version number and the low-byte contains the minor version number.

This property is read-only.

Data Type

Integer

See Also

Properties:

DeviceCount

Error Event

See Also Example

Description

Fires when an error occurs.

Syntax

Sub *ctlname*_**Error (***Error* **As Integer,** *ErrorMessage* **As String)**

Remarks

This event is fired whenever an error occurs. Both an error code and a textual description of the error are passed as arguments.

The argument *Error* holds the error number.

The argument *ErrorMessage* gives the error in string form.

See Also

Properties:

- Action (MIDI Input)
- Action (MIDI Output)

HasLRVolume Property

See Also Example

Description

Specifies whether or not the current device supports separate left and right volume control.

Usage

[*form.*][*control.*]**HasLRVolume**

Remarks

Specifies whether or not the current device (DeviceID) supports separate left and right volume control.

This property is read-only.

Data Type

Integer (boolean)

See Also

Properties:

HasVolume

HasVolume Property

See Also Example

Description

Specifies whether or not the current device supports volume.

Usage

[*form.*][*control.*]**HasVolume**

Remarks

Specifies whether or not the current device (DeviceID) supports volume.

This property is read-only.

Data Type

Integer (boolean)

See Also

Properties:

HasLRVolume

HMidiDevice Property

See Also

Description

Handle of MIDI device.

Usage

[*form.*][*control.*]**HMidiDevice**

Remarks

Device handle of MIDI device specified by DeviceID.   Only valid while device is open.

Data Type

Integer (long)

See Also

Properties:

- Action (MIDI input)
- Action (MIDI output)

ManufacturerID Property

See Also

Description

Manufacturer's ID for DeviceID.

Usage

[*form.*][*control.*]**ManufacturerID**

Remarks

This property returns the manufacturer's ID number for the device specified by DeviceID.

This property is read-only.

Data Type

Integer

See Also

Properties:

ProductID

MaxSysexSize Property

Description

Maximum amount of memory to allocate for receiving System Exclusive (Sysex) messages.

Usage

[form.][control.]**MaxSysexSize**[= *long*]

Remarks

This defaults to a value of zero (0).

This determines the amount of memory to allocate a buffer to store incoming Sysex messages.   If no system exclusive messages are going to be received, set this property to 0.

MaxSysexSize must be large enough to hold the largest system exclusive message that you anticipate receiving.   Sysex messages which do not fit into the buffer are discarded by MIDI drivers.

Data Type

Integer (long)

Message Event

See Also   Example

Description

Fires when a message is received.

Syntax

Sub *ctlname*_**Message ( )**

Remarks

This event is fired whenever MIDI messages are available and MessageEventEnable is set to True.

This is where all MIDI In data first arrives into Visual Basic. Examine the example source code in all of the Message event routines.

See Also

Properties:

- Action (MIDI Input)
- Action (MIDI Output)

Message Property

See Also Example

Description

Message byte.

Usage

[*form.*][*control.*]**Message**[= *integer*]

Remarks

Part of the data sent/received.

Data Type

Integer (0-255)

See Also

Properties:

Data1 and Data2

MessageCount Property

Example

Description

Number of messages available.

Usage

[*form.*][*control.*]**MessageCount**[= *integer*]

Remarks

As messages arrive at the MIDI Input control they are queued by the control. Your program can determine how many messages the MIDI Input control has queued by examining this property.

There is (or at least should be) an End of Track message at the end of each MIDI track. When you create a new track using the MIDI File control an End of Track message is placed in the track. The MessageCount property is actually one less than the number of messages since the End of Track message is not counted, cannot be accessed, and cannot be deleted.

Data Type

Integer (long)

MessageEventEnable Property

Description

Enables Message event.

Usage

[*form.*][*control.*]**MessageEventEnable**[= *boolean*]

Remarks

When this property is set to True, the Message event will be fired whenever messages are available.
When this property is set to False, the Message event will not be fired.

Data Type

Integer (boolean)

MessageSent Event

See Also Example

Description

Fires when a message is sent.

Syntax

Sub *ctlname*_**MessageSent (** *MessageTag* **As Long)**

Remarks

This event is fired what a tagged message has been sent to the MIDI channel. *MessageTag* identifies the message sent.

The MFPLAYR example utilizes the MessageSent event to trigger the VU indicators with MIDI Note On velocity data. This makes the VU meters react to the velocity data as the MIDI messages are sent from the MIDI Output queue.

See Also

Properties:

Action (MIDI Input)
Action (MIDI Output)

MessageTag Property

See Also Example

Description

The MessageTag property allows you to associate a long integer value with each particular MIDI message. When a MIDI message with a non-zero MessageTag is sent the MessageSent event will be fired.

Usage

[*form.*][*control.*]**MessageTag**[= *long*]

Remarks

Using the MessageTag property and MessageSent event you can sycnronize your program with MIDI events of your choosing.

Some of our sample programs use the MessageTag property to trigger the VU indicators with the velocity of each MIDI Note On event. The MessageTag property can be very useful in combination with the MessageSent event.

Data Type

Integer (long)

See Also

Events:

MessageSent

Notes Property

Description

Number of simultaneous notes the device may play.

Usage

[*form.*][*control.*]**Notes**

Remarks

Number of simultaneous notes (polyphony) that may be played by internal DeviceID.   Always zero for MIDI ports.

This property is read-only.

Data Type

Integer

ProductID Property

See Also

Description

Product ID for DeviceID.

Usage

[*form.*][*control.*]**ProductID**

Remarks

This property returns the product ID number for the device specified by DeviceID.

This property is read-only.

Data Type

Integer

See Also

Properties:

ManufacturerID

ProductName Property

Description

Product name for DeviceID.

Usage

[*form.*][*control.*]**ProductName**

Remarks

This property returns the product namefor the device specified by DeviceID.

This property is read-only.

Data Type

String

See Also

Properties:

DeviceID

QueueEmpty Event

See Also Example

Description

Fires when the output queue becomes empty.

Syntax

Sub *ctlname*_**QueueEmpty ( )**

Remarks

Fires when the output queue becomes empty.

Please note, there is a difference between the behavior of the VBX and the OCXes. The VBX continues to play after this event is fired. You may wish to stop the MIDI Output control once the queue becomes empty. Or, you may wish to queue up some more MIDI data.

The OCX controls stop playing when this message is fired. If you want to queue up more data, you will need to start playing again.

See Also

Properties:

- Action (MIDI Input)
- Action (MIDI Output)

State Property

SeeAlso Example

Description

Current state of DeviceID.

Usage

[*form.*][*control.*]**State**

Remarks

Setting this property returns the state of the MIDI device specified by DeviceID. The states are:

Constant	Value	Meaning
msMSClosed	0	Closed
msMSOpen	1	Open
msMSStarted	2	Started
msMSStopped	3	Stopped
msMSPaused	4	Paused

This property is read-only.

Data Type

Integer

See Also

Properties:

- Action (MIDI Input)
- Action (MIDI Output)

Time Property

Example

Description

Time of message in milliseconds.

Usage

[*form.*][*control.*]**Time**[= *integer*]

Remarks

Time of message in ticks.   It is important to note that Time has a different meaning in the MIDI input and output controls than it does in the MIDI file control.   MIDI input and output times are always milliseconds elapsed time since the start of either recording or playback, while the MIDI file control always sets Time to the number of Ticks which elapse between events.

For the MIDI input and MIDI output controls Time is always in milliseconds.

Data Type

Integer (long)

Timer Event

See Also Example

Description

Fires when a timer expires.

Syntax

Sub *ctlname*_**Timer ( )**

Remarks

Fires when a timer expires.

See Also

Properties:

Action (MIDI Input)
Action (MIDI Output)

Voices Property

Description

Number of voices supported by selected device.

Usage

[*form.*][*control.*]**Voices**

Remarks

Number of voices supported by internal MIDI (DeviceID).   Always zero for MIDI ports.

This property is read-only.

Data Type

Integer

VolumeLeft Property

Example

Description

Sets left side volume

Usage

[*form.*][*control.*]**VolumeLeft**[= *integer*]

Remarks

Sets the volume for the left channel of DeviceID. This value must range from 0 to 32767. If HasLRVolume is False, setting this property sets both VolumeLeft and VolumeRight.

You should save the VolumeRight and VolumeLeft properties when you open a MIDI device that supports volume control, and restore the properties just before you close the device. If you do not restore the properties the default volume for the MIDI device will be changed system-wide.

Data Type

Integer (0-32767)

VolumeRight Property

Example

Description

Sets right side volume

Usage

[*form.*][*control.*]**VolumeRight**[= *integer*]

Remarks

Sets the volume for the left channel of DeviceID. This value must range from 0 to 32767. If HasLRVolume is False, setting this property does nothing.

You should save the VolumeRight and VolumeLeft properties when you open a MIDI device that supports volume control, and restore the properties just before you close the device. If you do not restore the properties the default volume for the MIDI device will be changed system-wide.

Data Type

Integer (0-32767)

Action Property Example, MIDI Input Control

The following subroutine shows a sample MIDIInput_Message event handler. All of the available messages are read and output using the MIDI output control, this provides a MIDI-thru capability.

```
Sub MIDIInput1_Message()
 Dim Message As Integer
 Dim Data1   As Integer
 Dim Data2   As Integer

 Do While (MIDIInput1.MessageCount > 0 )
    '
    'This is the incoming MIDI data
    '
    Message = MIDIInput1.Message
    Data1 = MIDIInput1.Data1
    Data2 = MIDIInput1.Data2
    '
    ' Tell MIDIOutput1 to send the MIDI data
    '
    MIDIOutput1.Message = Message
    MIDIOutput1.Data1 = Data1
    MIDIOutput1.Data2 = Data2
    MIDIOutput1.Action = MIDIOUT_SEND
    '
    ' Remove the input message
    '
    MIDIInput1.Action = MIDIIN_REMOVE
 Loop
End Sub
```

Close Copy Print

Action Property Example, MIDI Output Control

The following subroutine shows a sample MIDIInput_Message event handler. All of the available messages are read and output using the MIDI output control, this provides a MIDI-thru capability.

```
Sub MIDIInput1_Message()
  Dim Message As Integer
  Dim Data1   As Integer
  Dim Data2   As Integer

  Do While (MIDIInput1.MessageCount > 0 )
      '
      'This is the incoming MIDI data
      '
      Message = MIDIInput1.Message
      Data1 = MIDIInput1.Data1
      Data2 = MIDIInput1.Data2
      '
      ' Tell MIDIOutput1 to send the MIDI data
      '
      MIDIOutput1.Message = Message
      MIDIOutput1.Data1 = Data1
      MIDIOutput1.Data2 = Data2
      MIDIOutput1.Action = MIDIOUT_SEND
      '
      ' Remove the input message
      '
      MIDIInput1.Action = MIDIIN_REMOVE
  Loop
End Sub
```

Buffer Property Example

In this example, a Sysex message is sent which resets the Roland SoundCanvas SC-88 to General Midi mode.

```
Sub SetGMMode_Click ()
        Midioutput1.Buffer = Chr$(&HF0) + Chr$(&H7E) + Chr$(&H7F) + Chr$(9) + Chr$(1) + Chr$
(&HF7)
        Midioutput1.Message = &HF0
        Midioutput1.Action = MIDIOUT_SEND
End Sub
```

In this example the first and last bytes (&HF0 and &HF7) signal the beginning and end of a Sysex message. The middle bytes are the Sysex messages contents.

CanCache Property Example

Close Copy Print

Data1 and Data2 Properties Example

The following subroutine shows a sample MIDIInput_Message event handler.   All of the available messages are read and output using the MIDI output control,   this provides a MIDI-thru capability.

```
Sub MIDIInput1_Message()
 Dim Message As Integer
 Dim Data1   As Integer
 Dim Data2   As Integer

 Do While (MIDIInput1.MessageCount > 0 )
    '
    'This is the incoming MIDI data
    '
    Message = MIDIInput1.Message
    Data1 = MIDIInput1.Data1
    Data2 = MIDIInput1.Data2
    '
    ' Tell MIDIOutput1 to send the MIDI data
    '
    MIDIOutput1.Message = Message
    MIDIOutput1.Data1 = Data1
    MIDIOutput1.Data2 = Data2
    MIDIOutput1.Action = MIDIOUT_SEND
    '
    ' Remove the input message
    '
    MIDIInput1.Action = MIDIIN_REMOVE
 Loop
End Sub
```

DeviceCount Property Example

This example shows how to load combo-boxes with lists of input devices and output devices.

```
Sub Form_Load ()
 Dim i As Integer

 '
 ' Fill output device combo box
 '
 For i = -1 To MIDIOutput1.DeviceCount - 1
    MIDIOutput1.DeviceID = i
    OutputDevCombo.AddItem MIDIOutput1.ProductName
 Next
 '
 ' Select first in list
 '
 MIDIOutput1.DeviceID = -1
 OutputDevCombo.ListIndex = 0
 '
 ' Fill input device combo box
 '
 For i = 0 To MIDIInput1.DeviceCount - 1
    MIDIInput1.DeviceID = i
    InputDevCombo.AddItem MIDIInput1.ProductName
 Next
 '
 ' Select first in list
 '
 MIDIInput1.DeviceID = -1
 InputDevCombo.ListIndex = 0
End Sub
```

Close Copy Print

DeviceID Property Example

This example shows how to load combo-boxes with lists of input devices and output devices.

```
Sub Form_Load ()
 Dim i As Integer

 '
 ' Fill output device combo box
 '
 For i = -1 To MIDIOutput1.DeviceCount - 1
    MIDIOutput1.DeviceID = i
    OutputDevCombo.AddItem MIDIOutput1.ProductName
 Next
 '
 ' Select first in list
 '
 MIDIOutput1.DeviceID = -1
 OutputDevCombo.ListIndex = 0
 '
 ' Fill input device combo box
 '
 For i = 0 To MIDIInput1.DeviceCount - 1
    MIDIInput1.DeviceID = i
    InputDevCombo.AddItem MIDIInput1.ProductName
 Next
 '
 ' Select first in list
 '
 MIDIInput1.DeviceID = -1
 InputDevCombo.ListIndex = 0
End Sub
```

Close Copy Print

DeviceType Property Example

Close Copy Print

DriverVersion Property Example

Error Event Example

```
Sub MIDIOutput1_Error (ErrorCode As Integer, ErrorMessage As String)
      MsgBox ErrorMessage
End Sub
```

Error Event Example

```
Sub MIDIOutput1_Error (ErrorCode As Integer, ErrorMessage As String)
      MsgBox ErrorMessage
End Sub
```

Close Copy Print

HasLRVolume Property Example

```
 Sub CloseOutputDevice ()
   '
   ' Restore volume before closing
   '
   If MIDIOutput1.State >= MIDISTATE_OPEN Then
      If (MIDIOutput1.HasLRVolume) Then
         MIDIOutput1.VolumeLeft = lVolume
         MIDIOutput1.VolumeRight = rVolume
      ElseIf (MIDIOutput1.HasVolume) Then
         MIDIOutput1.VolumeLeft = lVolume
      End If
      '
      ' Close
      '
   MIDIOutput1.Action = MIDIOUT_CLOSE
   End If
 End Sub
```

HasVolume Property Example

```
Sub CloseOutputDevice ()
  '
  ' Restore volume before closing
  '
  If MIDIOutput1.State >= MIDISTATE_OPEN Then
     If (MIDIOutput1.HasLRVolume) Then
        MIDIOutput1.VolumeLeft = lVolume
        MIDIOutput1.VolumeRight = rVolume
     ElseIf (MIDIOutput1.HasVolume) Then
        MIDIOutput1.VolumeLeft = lVolume
     End If
     '
     ' Close
     '
  MIDIOutput1.Action = MIDIOUT_CLOSE
  End If
End Sub
```

Close Copy Print

HMidiDevice Property Example

ManufacturerID Property Example

Close Copy Print

Message Event Example

```
 Sub MIDIInput1_Message ()
   Dim InMessage As Integer
   Dim InData1 As Integer
   Dim InData2 As Integer
   Dim Y As Integer

   If (fGotFirst = False) Then
      PreviousTime = MIDIInput1.Time
      fGotFirst = True
      fRecording = True
   End If
   '
   'This do while loop allows you to take all the messages that are
   'waiting in the message queue.
   '
   Do While MIDIInput1.MessageCount > 0
      '
      'This is the incoming MIDI data
      '
      InMessage = MIDIInput1.Message
      InData1 = MIDIInput1.Data1
      InData2 = MIDIInput1.Data2
      '
      ' Copy input to output?
      '
      If (MidiThruCheck.Value) Then
         '
         'Tell MIDIOutput1 to send the MIDI data
         '
         MIDIOutput1.Message = InMessage
         MIDIOutput1.Data1 = InData1
         MIDIOutput1.Data2 = InData2
         MIDIOutput1.Action = MIDIOUT_SEND
      End If

      If (InsertRecordingCheck.Value) Then
         '
         ' Copy message parameters
         '
         MIDIFile1.Message = MIDIOutput1.Message
         MIDIFile1.Data1 = MIDIOutput1.Data1
         MIDIFile1.Data2 = MIDIOutput1.Data2
         '
         ' Calculate time in ticks
         '
         CurrentTime = MIDIInput1.Time
         MIDIFile1.Time = Int(CurrentTime - PreviousTime) * ticksPerMs
         PreviousTime = CurrentTime
         '
         ' insert message into MIDI file
         '
         MIDIFile1.Action = MIDIFILE_INSERT_MESSAGE
      End If
      '
      'Remove the MIDI data from the MIDI IN queue
      '
      MIDIInput1.Action = MIDIIN_REMOVE
  Loop
End Sub
```

Close Copy Print

Message Property Example

The following subroutine shows a sample MIDIInput_Message event handler.   All of the available messages are read and output using the MIDI output control,   this provides a MIDI-thru capability.

```
Sub MIDIInput1_Message()
  Dim Message As Integer
  Dim Data1   As Integer
  Dim Data2   As Integer

  Do While (MIDIInput1.MessageCount > 0 )
      '
      'This is the incoming MIDI data
      '
      Message = MIDIInput1.Message
      Data1 = MIDIInput1.Data1
      Data2 = MIDIInput1.Data2
      '
      ' Tell MIDIOutput1 to send the MIDI data
      '
      MIDIOutput1.Message = Message
      MIDIOutput1.Data1 = Data1
      MIDIOutput1.Data2 = Data2
      MIDIOutput1.Action = MIDIOUT_SEND
      '
      ' Remove the input message
      '
      MIDIInput1.Action = MIDIIN_REMOVE
  Loop
End Sub
```

MessageCount Property Example

The following subroutine shows a sample MIDIInput_Message event handler.   All of the available messages are read and output using the MIDI output control,   this provides a MIDI-thru capability.

```
Sub MIDIInput1_Message()
 Dim Message As Integer
 Dim Data1   As Integer
 Dim Data2   As Integer

 Do While (MIDIInput1.MessageCount > 0 )
    '
    'This is the incoming MIDI data
    '
    Message = MIDIInput1.Message
    Data1 = MIDIInput1.Data1
    Data2 = MIDIInput1.Data2
    '
    ' Tell MIDIOutput1 to send the MIDI data
    '
    MIDIOutput1.Message = Message
    MIDIOutput1.Data1 = Data1
    MIDIOutput1.Data2 = Data2
    MIDIOutput1.Action = MIDIOUT_SEND
    '
    ' Remove the input message
    '
    MIDIInput1.Action = MIDIIN_REMOVE
 Loop
End Sub
```

MessageEventEnable Property Example

Close Copy Print

MessageSent Event Example

```
Sub MIDIOutput1_MessageSent (MessageTag As Long)
  If (MessageTag = 1) Then
      Shape1.Visible = True
  Else
      Shape1.Visible = False
  End If
End Sub
```

Close Copy Print

MessageTag Property Example

```
Sub MIDIOutput1_MessageSent (MessageTag As Long)
  If (MessageTag = 1) Then
     Shape1.Visible = True
  Else
     Shape1.Visible = False
  End If
End Sub
```

Close Copy Print

Notes Property Example

Close Copy Print

ProductID Property Example

ProductName Property Example

This example shows how to load combo-boxes with lists of input devices and output devices.

```
Sub Form_Load ()
 Dim i As Integer

 '
 ' Fill output device combo box
 '
 For i = -1 To MIDIOutput1.DeviceCount - 1
    MIDIOutput1.DeviceID = i
    OutputDevCombo.AddItem MIDIOutput1.ProductName
 Next
 '
 ' Select first in list
 '
 MIDIOutput1.DeviceID = -1
 OutputDevCombo.ListIndex = 0
 '
 ' Fill input device combo box
 '
 For i = 0 To MIDIInput1.DeviceCount - 1
    MIDIInput1.DeviceID = i
    InputDevCombo.AddItem MIDIInput1.ProductName
 Next
 '
 ' Select first in list
 '
 MIDIInput1.DeviceID = -1
 InputDevCombo.ListIndex = 0
End Sub
```

Close Copy Print

State Property Example

This example checks the MIDIOutput State property to see if the output device is open before trying to close it.

```
Sub CloseOutputDevice ()
  '
  ' Restore volume before closing
  '
  If MIDIOutput1.State >= MIDISTATE_OPEN Then
     If (MIDIOutput1.HasLRVolume) Then
        MIDIOutput1.VolumeLeft = lVolume
        MIDIOutput1.VolumeRight = rVolume
     ElseIf (MIDIOutput1.HasVolume) Then
        MIDIOutput1.VolumeLeft = lVolume
     End If
     '
     ' Close
     '
     MIDIOutput1.Action = MIDIOUT_CLOSE
  End If
End Sub
```

Time Property Example

This example shows how to change time for the current message.

```
Sub CmdModifyMessageTime_Click ()
 MIDIFile1.Time = Val(TimeEdit.Text)
 MIDIFile1.Action = MIDIFILE_MODIFY_MESSAGE
End Sub
```

Close Copy Print

Timer Event Example

```
Sub MIDIOutput1_Timer (TimerTag As Long)
 If (TimerTag = 1) Then
    Shape1.Visible = True
 Else
    Shape1.Visible = False
 End If
End Sub
```

Close Copy Print

Voices Property Example

VolumeLeft Property Example

```
Sub CloseOutputDevice ()
  '
  ' Restore volume before closing
  '
  If MIDIOutput1.State >= MIDISTATE_OPEN Then
     If (MIDIOutput1.HasLRVolume) Then
        MIDIOutput1.VolumeLeft = lVolume
        MIDIOutput1.VolumeRight = rVolume
     ElseIf (MIDIOutput1.HasVolume) Then
        MIDIOutput1.VolumeLeft = lVolume
     End If
     '
     ' Close
     '
  MIDIOutput1.Action = MIDIOUT_CLOSE
  End If
End Sub
```

Close Copy Print

VolumeRight Property Example

```
Sub CloseOutputDevice ()
  '
  ' Restore volume before closing
  '
  If MIDIOutput1.State >= MIDISTATE_OPEN Then
     If (MIDIOutput1.HasLRVolume) Then
        MIDIOutput1.VolumeLeft = lVolume
        MIDIOutput1.VolumeRight = rVolume
     ElseIf (MIDIOutput1.HasVolume) Then
        MIDIOutput1.VolumeLeft = lVolume
     End If
     '
     ' Close
     '
  MIDIOutput1.Action = MIDIOUT_CLOSE
  End If
End Sub
```

Getting Custom Controls Written

If you or your organization would like to have custom controls written, you can contact us at the following:

Mabry Software, Inc.
Post Office Box 31926
Seattle, WA   98103-1926

Phone: 206-634-1443
Fax: 206-632-0272

CompuServe: 71231,2066
Internet: mabry@mabry.com

You can also contact Zane Thomas.   He can be reached at:

Zane Thomas
Post Office Box 121
Indianola, WA   98342

Internet: zane@mabry.com

Licensing Information

Legalese Version

Mabry Software grants a license to use the enclosed software to the original purchaser. Copies may be made for back-up purposes only. Copies made for any other purpose are expressly prohibited, and adherence to this requirement is the sole responsibility of the purchaser.

Customer written executable applications containing embedded Mabry products may be freely distributed, without royalty payments to Mabry Software, provided that such distributed Mabry product is bound into these applications in such a way so as to prohibit separate use in design mode, and that such Mabry product is distributed only in conjunction with the customers own software product. The Mabry Software product may not be distributed by itself in any form.

Neither source code for Mabry Software products nor modified source code for Mabry Software products may be distributed under any circumstances, nor may you distribute .OBJ, .LIB, etc. files that contain our routines. This control may be used as a constituent control only if the compound control thus created is distributed with and as an integral part of an application. Permission to use this control as a constituent control does not grant a right to distribute the license (LIC) file or any other file other than the control executable itself.This license may be transferred to a third party only if all existing copies of the software and its documentation are also transferred.

This product is licensed for use by only one developer at a time. Mabry Software expressly prohibits installing this product on more than one computer if there is any chance that both copies will be used simultaneously. This restriction also extends to installation on a network server, if more than one workstation will be accessing the product. All developers working on a project which includes a Mabry Software product, even though not working directly with the Mabry product, are required to purchase a license for that Mabry product.

This software is provided as is. Mabry Software makes no warranty, expressed or implied, with regard to the software. All implied warranties, including the warranties of merchantability and fitness for a particular use, are hereby excluded.

MABRY SOFTWARE'S LIABILITY IS LIMITED TO THE PURCHASE PRICE. Under no circumstances shall Mabry Software or the authors of this product be liable for any incidental or consequential damages, nor for any damages in excess of the original purchase price.

To be eligible for free technical support by telephone, the Internet, CompuServe, etc. and to ensure that you are notified of any future updates, please complete the enclosed registration card and return it to Mabry Software.

English Version

We require that you purchase one copy of a control per developer on a project. If this is met, you may distribute the control with your application royalty free. You may never distribute the LIC file. You may not change the product in any way that removes or changes the requirement of a license file.

We encourage the use of our controls as constituent controls when the compound controls you create are an integral part of your application. But we don't allow distribution of our controls as constituents of other controls when the compound control is not part of an application. The reason we need to have this restriction is that without it someone might decide to use our control as a constituent, add some trivial (or even non-trivial) enhancements and then sell the compound control. Obviously there would be little difference between that and just plain reselling our control.

If you have purchased the source code, you may not re-distribute the source code either (nor may you copy it into your own project). Mabry Software retains the copyright to the source code.

Your license is transferable. The original purchaser of the product must make the transfer request. Contact us for further information.