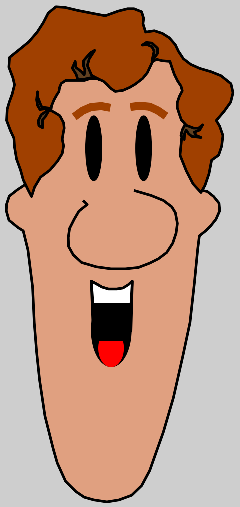
How to get started with VB and MIDI

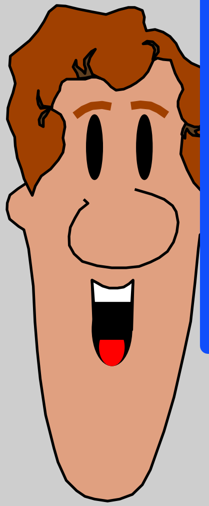
Hi!
My name is Gaute
I will guide you through
your first steps of using
Visual Basic to program
MIDI using the
MMSYSTEM.DLL



The graphic in this presentation will look best on a VGA (640x480) screen

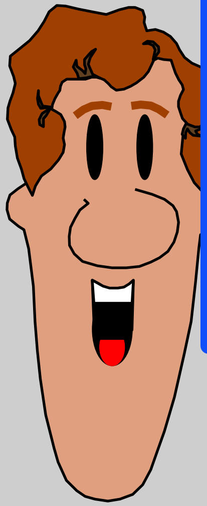
To fully follow me in this presentation
it would be best if you where familiar with
MIDI messages and
the technique of calling DLL's from VB.

If not
Read some VB documentation,
look for declare in the VB on-line help
and read about MIDI implementation in your
ROLAND owners manual.



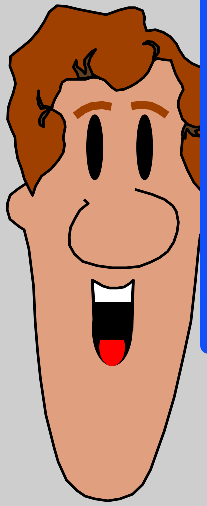
anyway you will learn something.....

First we will talk a little theory and then I will guide you through a simple application to demonstrate some of the commands and techniques.



MMSYSTEM.DLL is the low-level entry to the Multimedia for Windows.
It contains routines for playing video, music reading the joystick etc..
I will focus on the routines for MIDI only.

Since the routines are low-level you may find them cumbersome to use
but they give us very good control.

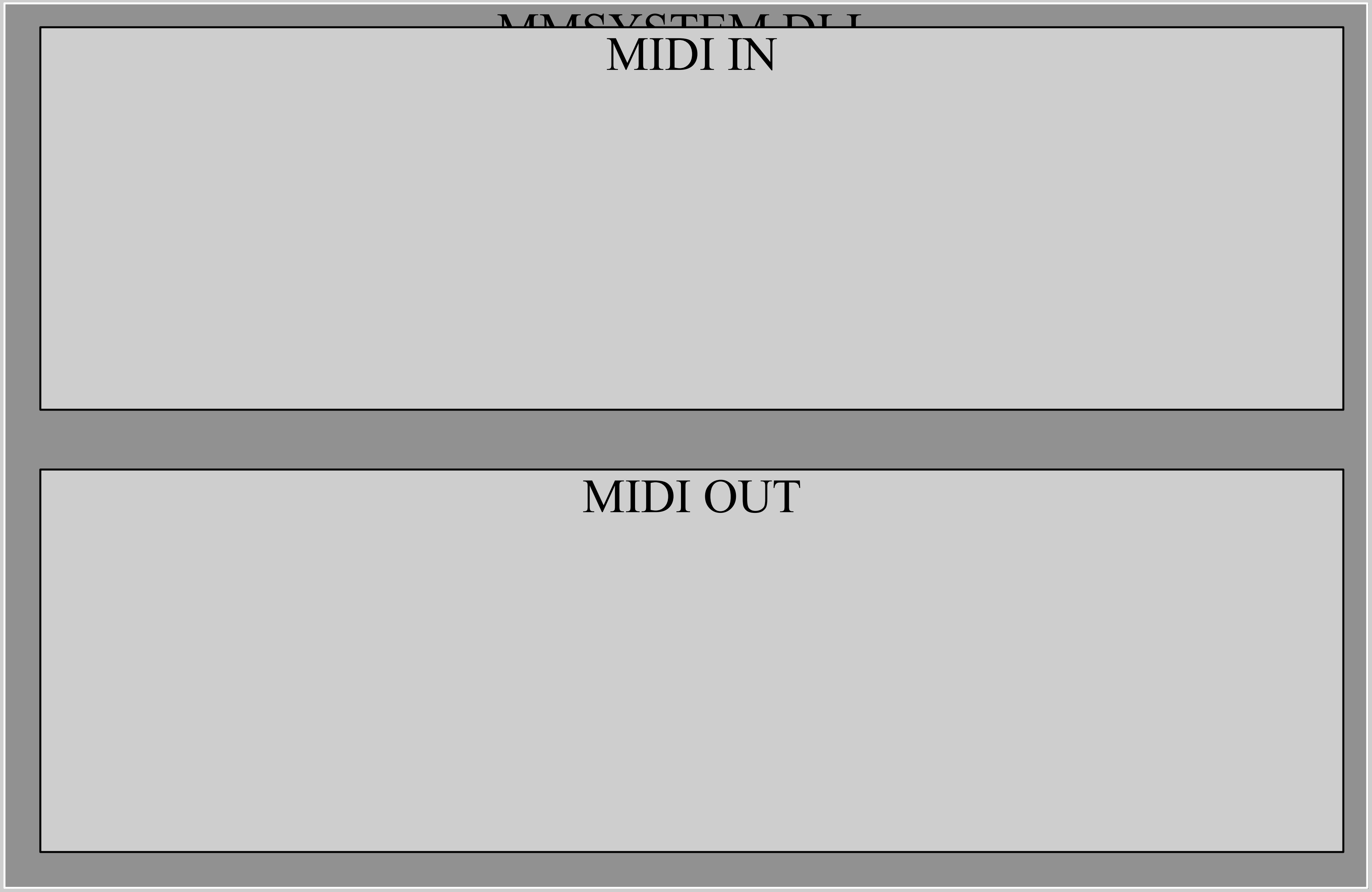


Overview of MMSYSTEM.DLL

MMSYSTEM.DLL
(MIDI subset)

Can be divided in to parts..

Overview of MMSYSTEM.DLL



Overview of MMSYSTEM.DLL

MMSYSTEM.DLL

MIDI IN

midiOutGetNumDevs
midiOutGetDevCaps

midiOutOpen
midiOutClose

midiOutGetErrorText

MIDI OUT

midiInGetNumDevs
midiInGetDevCaps

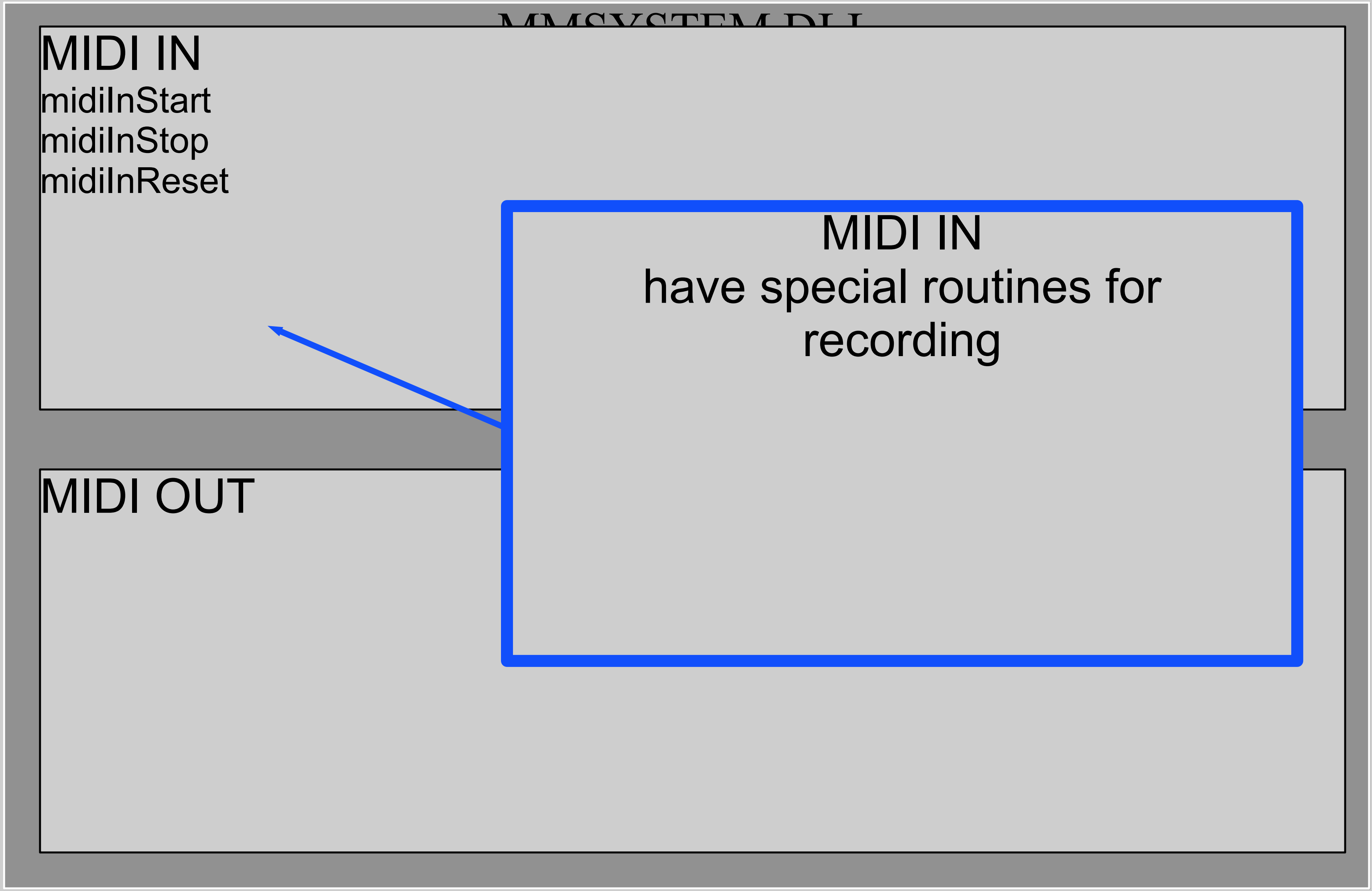
midiInOpen
midiInClose

midiInGetErrorText

Both parts have routines to get;

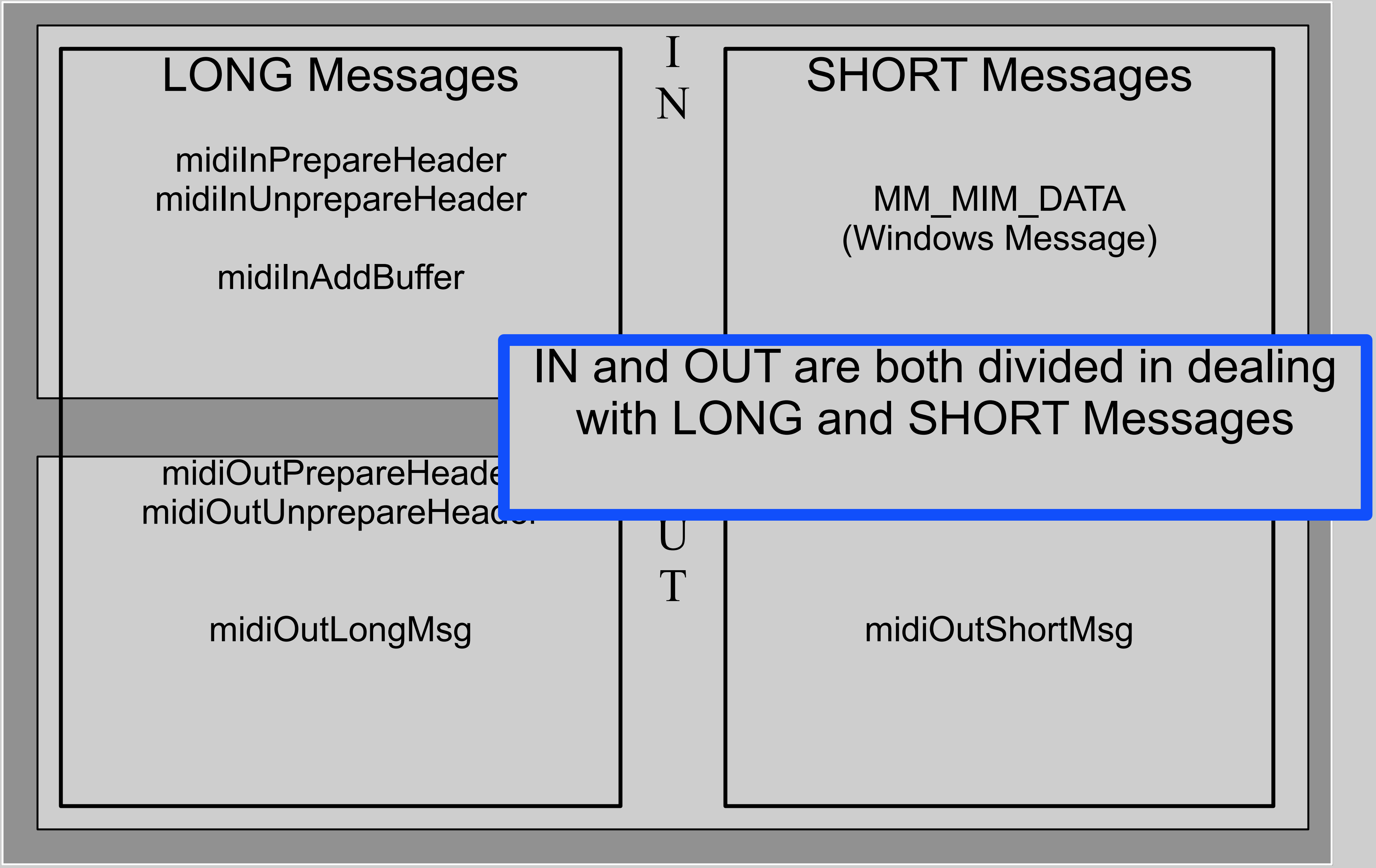
number of installed devices
and capabilities of those devices
and
Opening and closing
of the devices.

Overview of MMSYSTEM.DLL



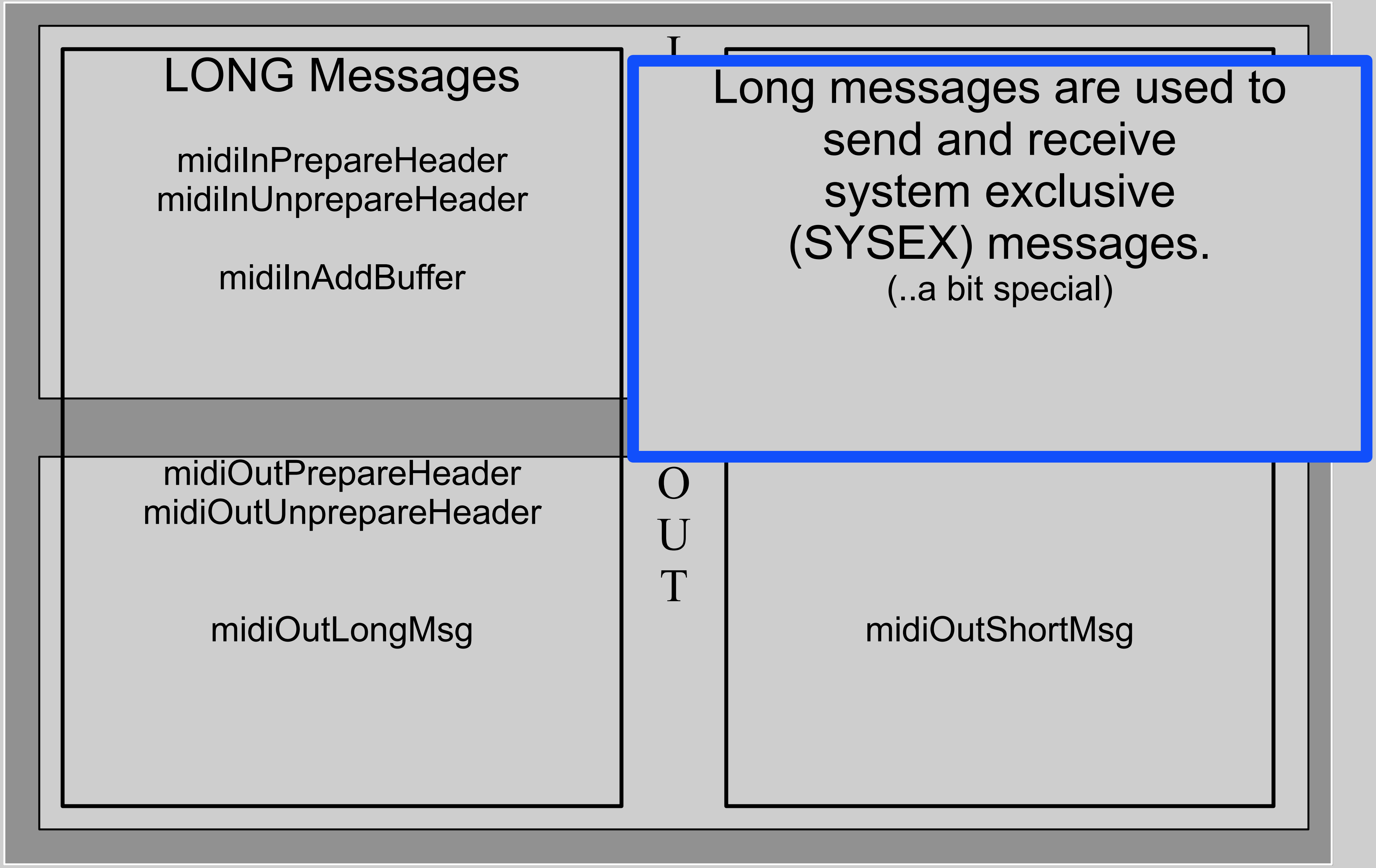
MIDI IN
have special routines for recording

Overview of MMSYSTEM.DLL



IN and OUT are both divided in dealing with LONG and SHORT Messages

Overview of MMSYSTEM.DLL



Long messages are used to send and receive system exclusive (SYSEX) messages.
(..a bit special)

Overview of MMSYSTEM.DLL

I
N

LONG Messages

midiInPrepareHeader
midiInUnprepareHeader

midiInAddBuffer

SHORT Messages

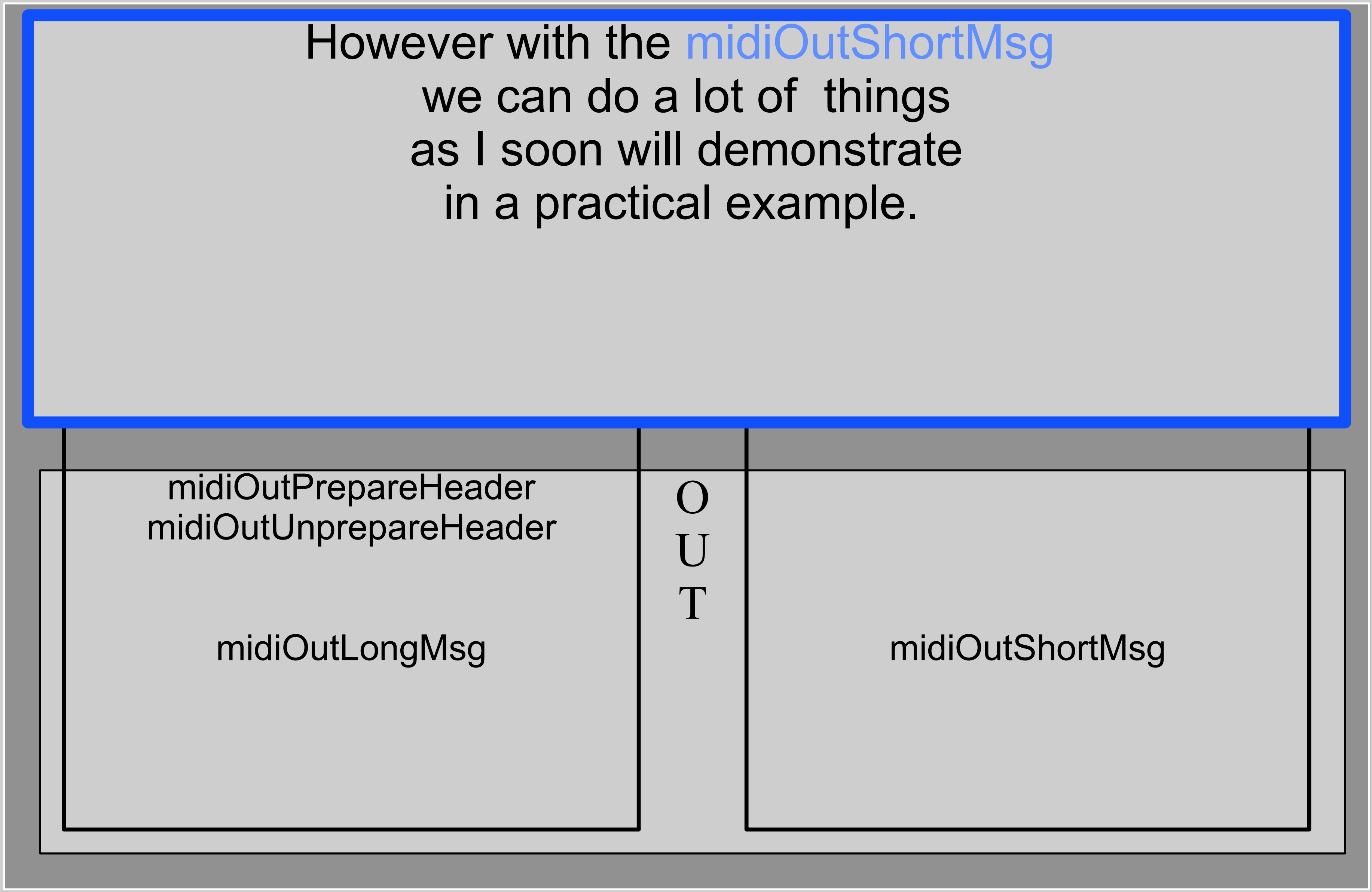
MM_MIM_DATA
(Windows Message)

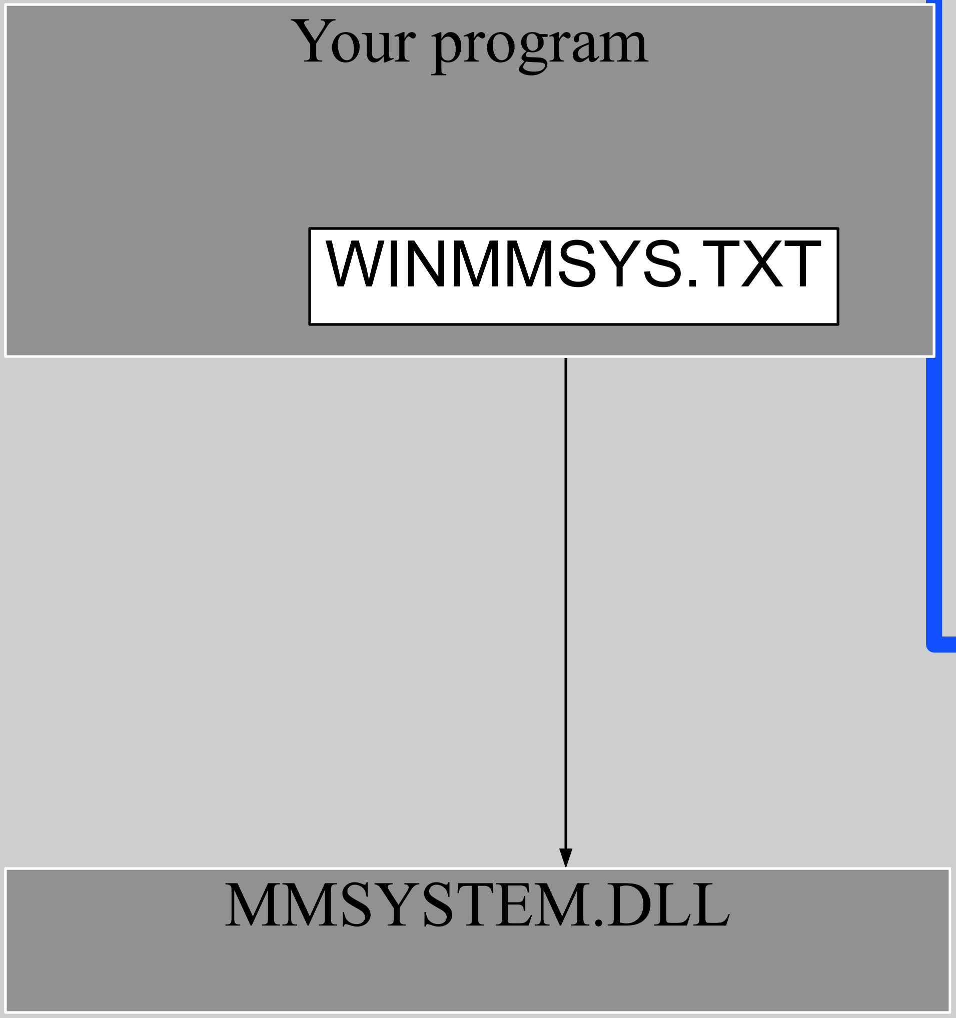
MIDI input can be done with call-back or messages.
Call-back is next to impossible in VB.
Trapping Windows Messages in VB requires a special VBX.

(Like the “MessageBlaster” in
the VB\MSGBLAST directory on this CD-ROM)

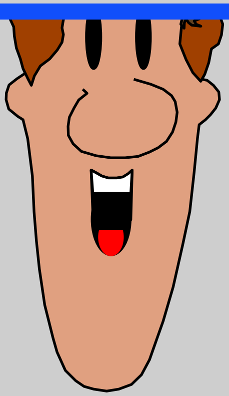
Overview of MMSYSTEM.DLL

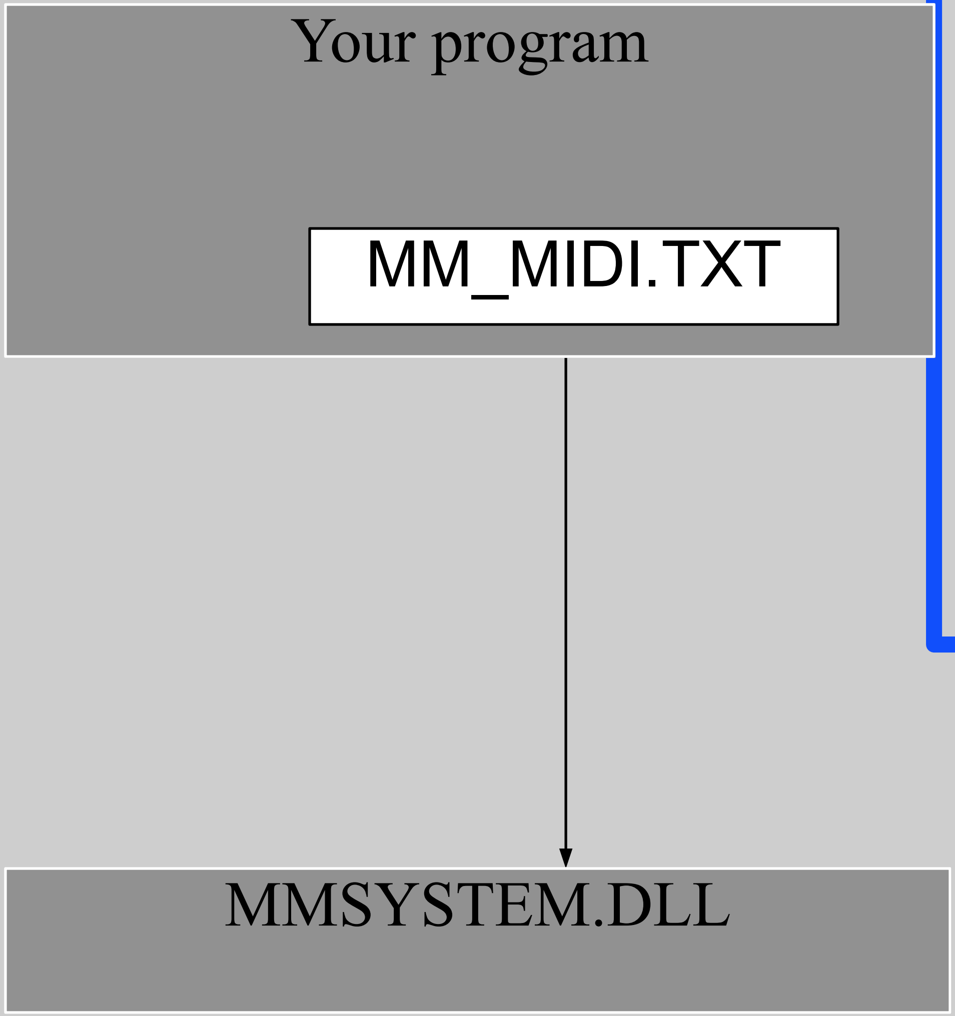
However with the midiOutShortMsg
we can do a lot of things
as I soon will demonstrate
in a practical example.



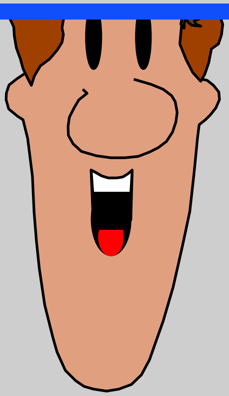


By adding the definitions
contained in the file
WINMMSYS.TXT
that comes with VB
you can call the
MMSYSTEM.DLL
directly from your program.

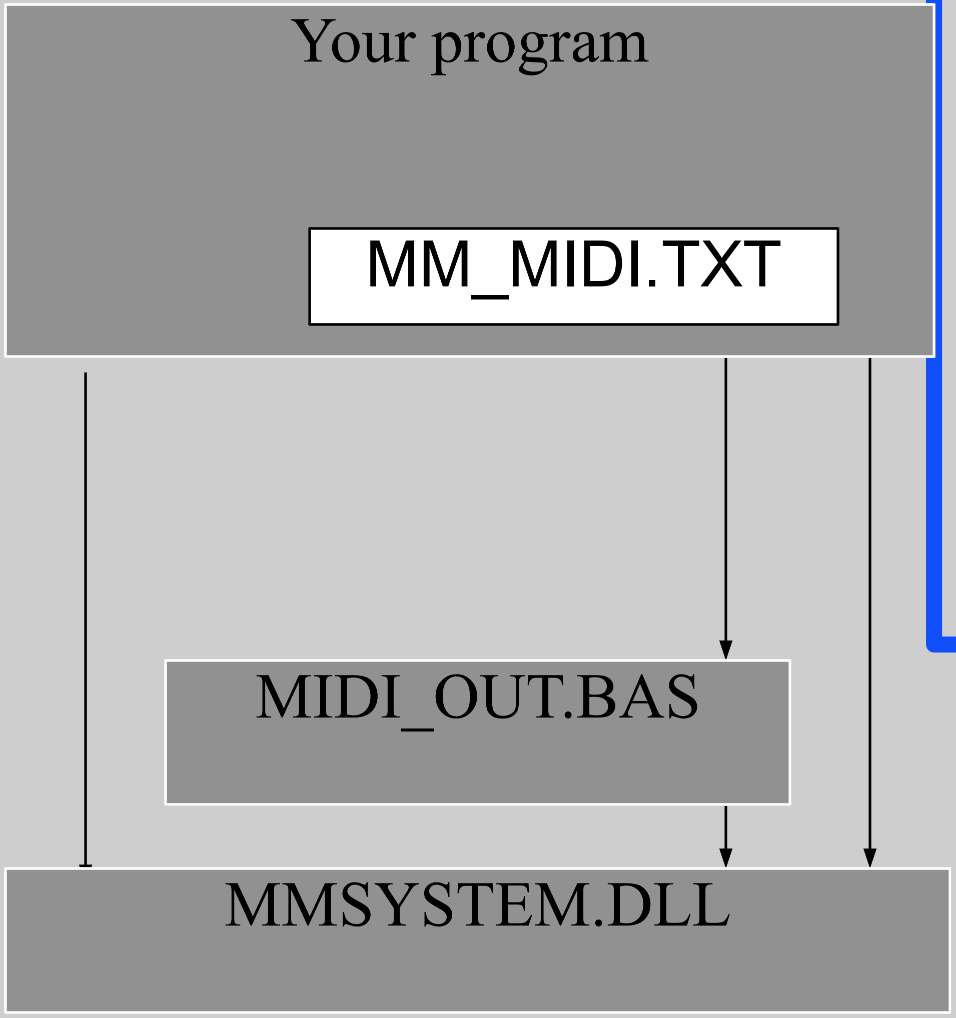


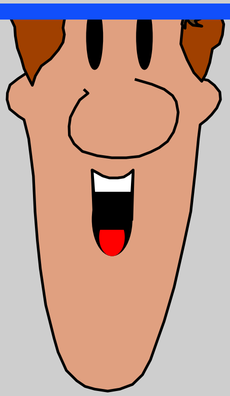


WINMMSYS.TXT
Is quite big so I have made a "MIDI only" version called MM_MIDI.TXT and placed it in the MMSYSTEM\VB_LIBS directory on this CD-ROM.

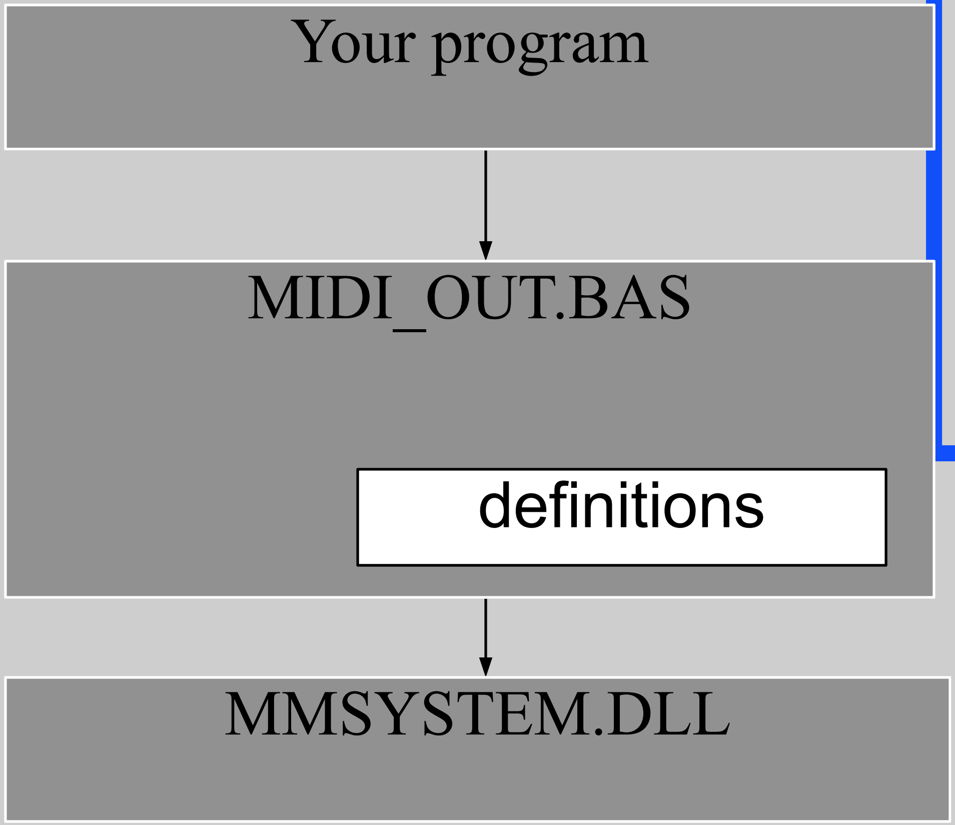


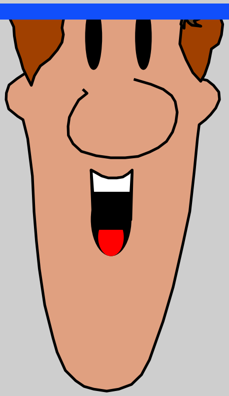
To make things a bit easier
I have wrapped the more
important routines in the
MMSYSTEM.DLL
in VB code and placed them
in a VB module file
called MIDI_OUT.BAS
in the same directory.



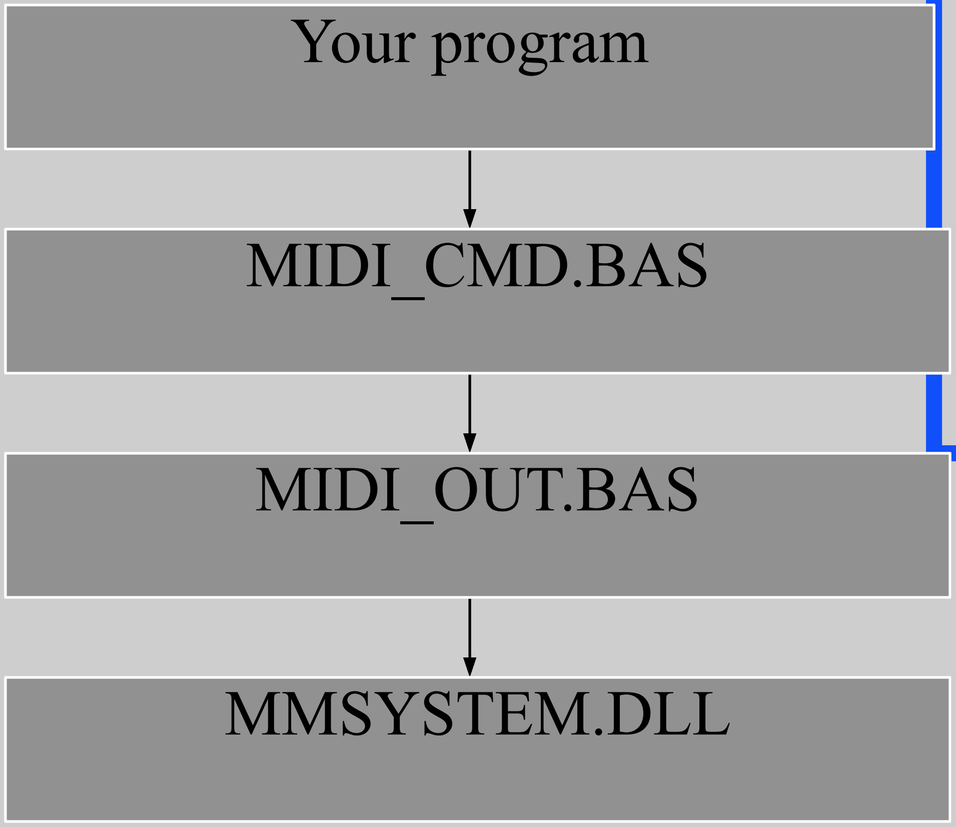


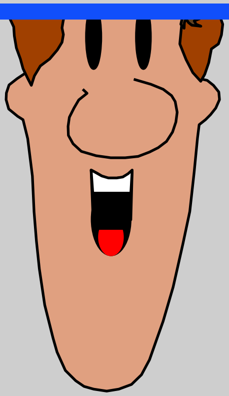
If you only call
MIDI_OUT.BAS
it is more practical to let this
module be self supplied
with definitions
so MIDI_OUT.BAS
contains all necessary
definitions.





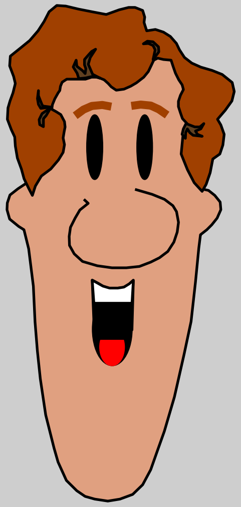
Some of the more common MIDI commands
I have put in VB routines and placed in a VB file called
MIDI_CMD.BAS
The file is in the
MMSYSTEM\VB_LIBS
directory on the CD.

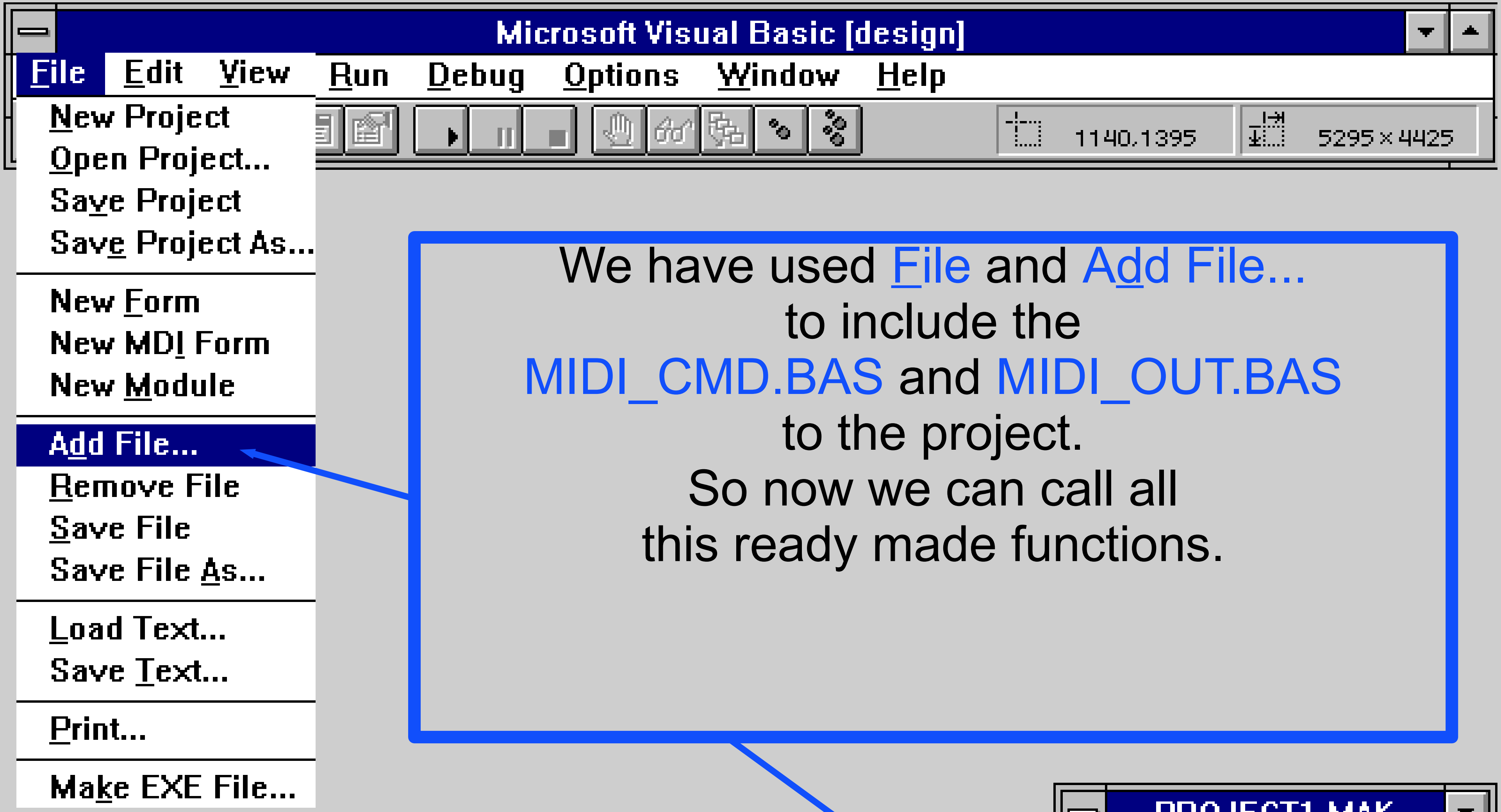




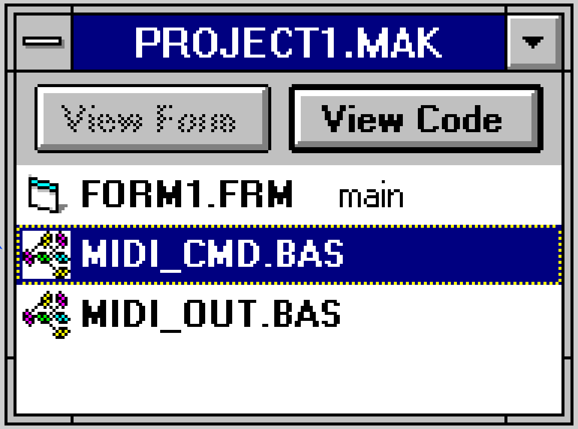
..but enough talking!
Let us start VB
and get to some code.

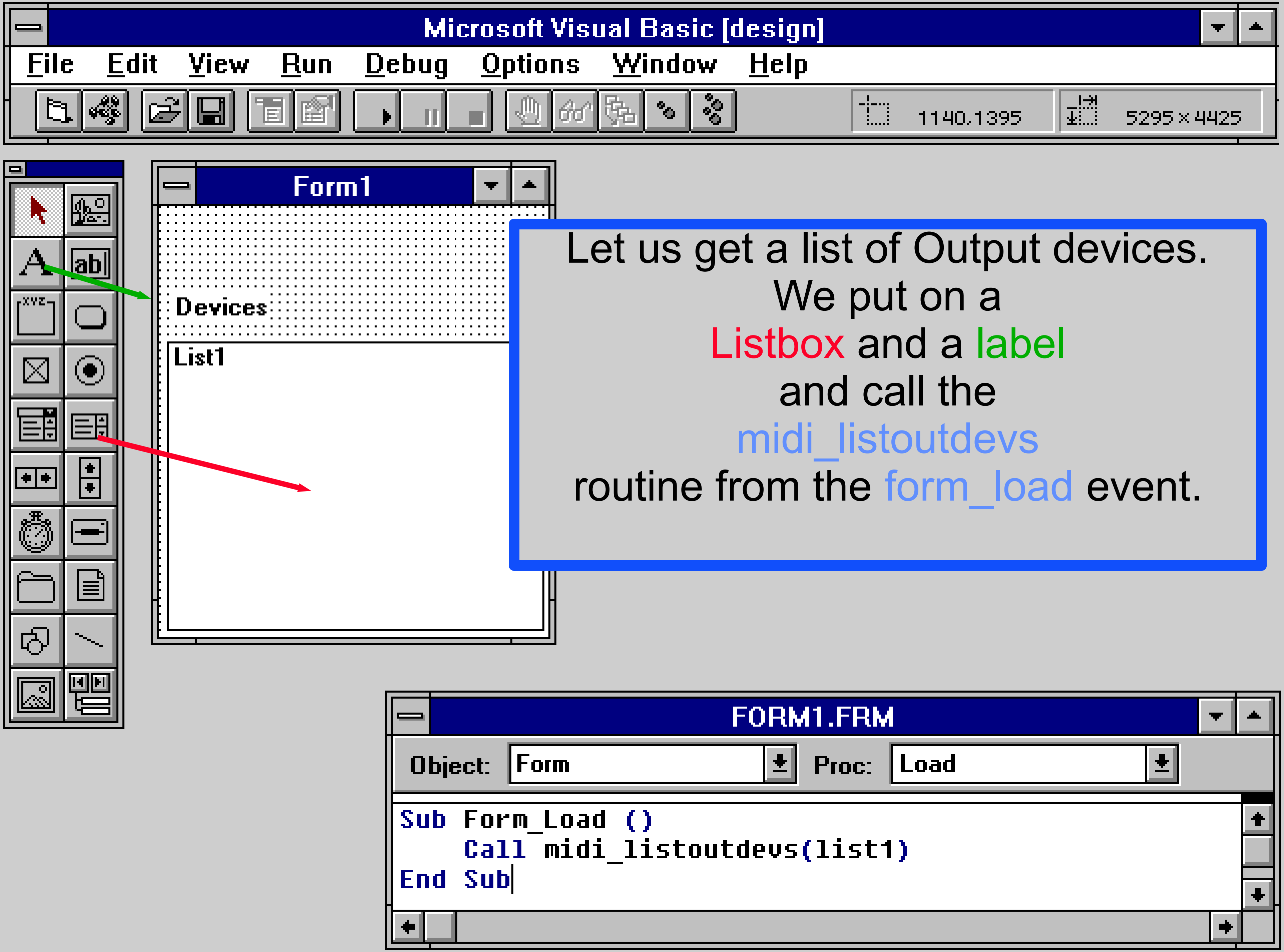
I will not go in detail the same way
as I did with the MCI.
I will assume that you know VB
and how to use it as a programming tool.





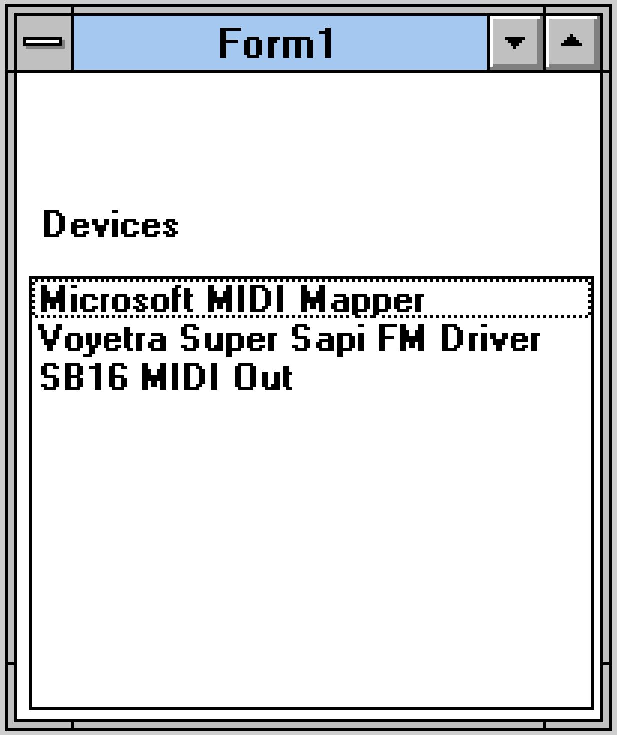
We have used File and Add File...
to include the
MIDI_CMD.BAS and MIDI_OUT.BAS
to the project.
So now we can call all
this ready made functions.




Microsoft Visual Basic [design]
File Edit View Run Debug Options Window Help
1140,1395
5295 x 4425
Form1
Devices
List1
FORM1.FRM
Object: Form
Proc: Load

Let us get a list of Output devices.
We put on a
Listbox and a label
and call the
midi_listoutdevs
routine from the form_load event.

```
Sub Form_Load ()
    Call midi_listoutdevs(list1)
End Sub
```


Form1

Devices

Microsoft MIDI Mapper
Voyetra Super Sapi FM Driver
SB16 MIDI Out

On a machine with a SB16 card
and a external ROLAND sound module
this will look something like this:

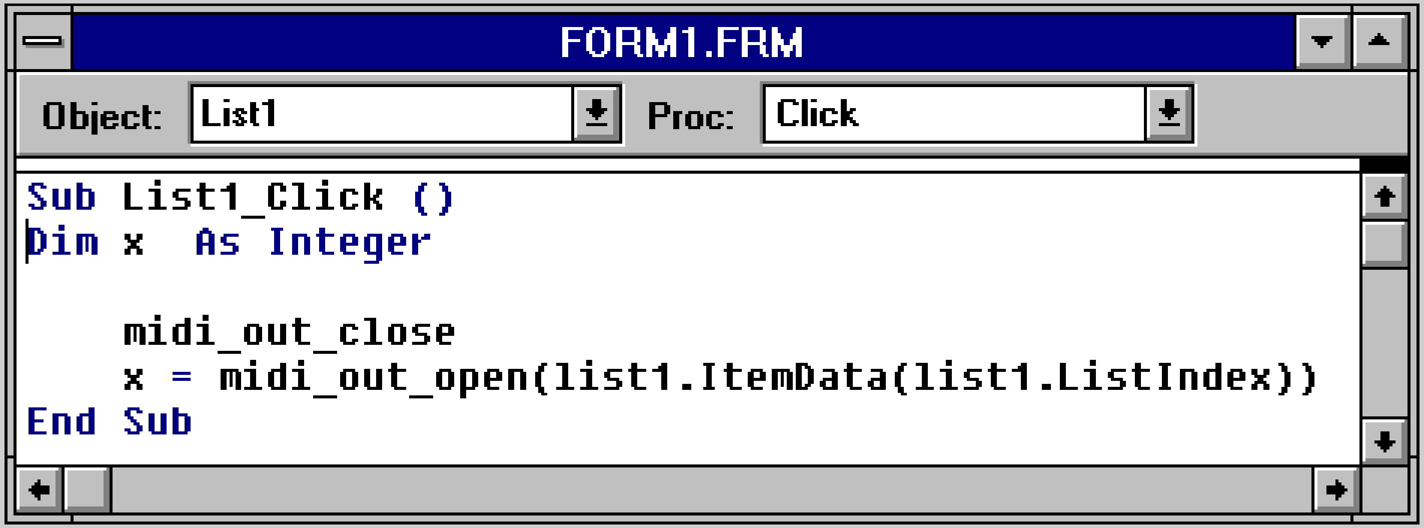
If we pick the MIDI mapper
the output will go where the map direct it

The Voyetra will play on the internal FM
synth (so we can compare the sound)

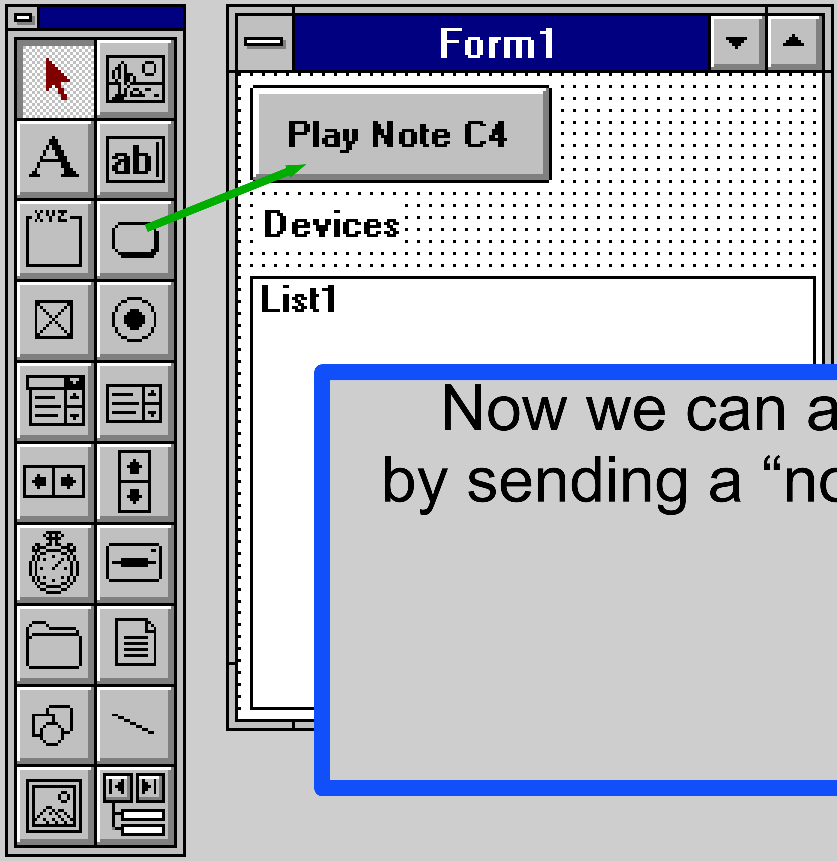
and the SB16 will send the notes
directly to the external box.

To make the user choose the device to open, we add code on the click event of the list box.

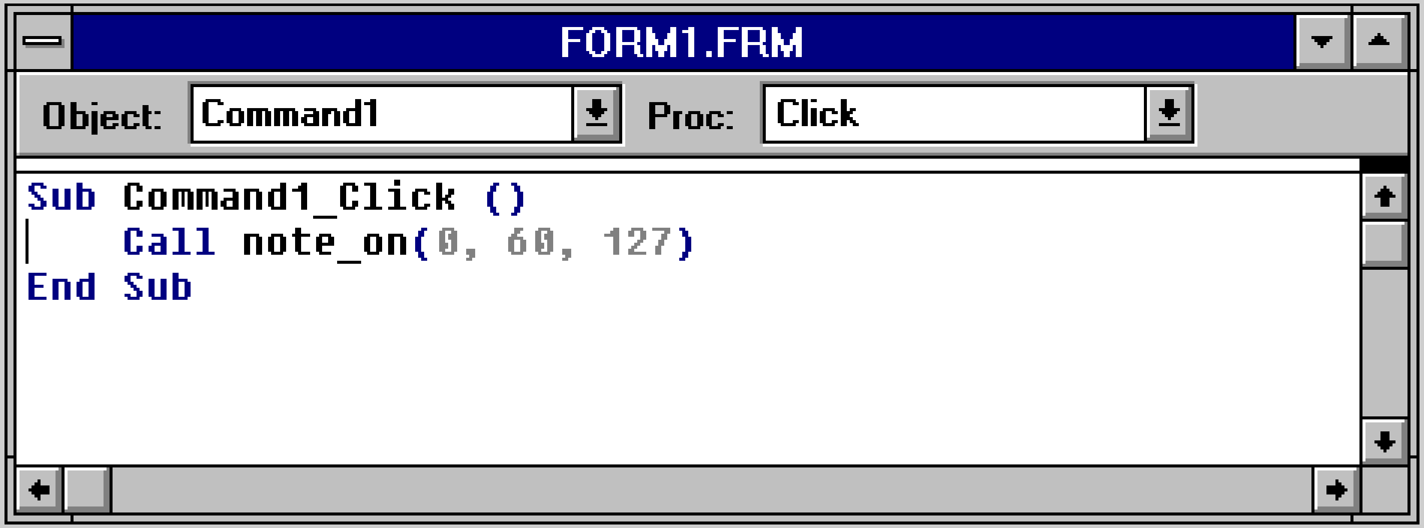
Here we open the device that the user selects by sending the related device ID that where stored in the ItemData of the listbox by the midi_listoutdevs routine.



```
Sub List1_Click ()
Dim x  As Integer

    midi_out_close
    x = midi_out_open(list1.ItemData(list1.ListIndex))
End Sub
```

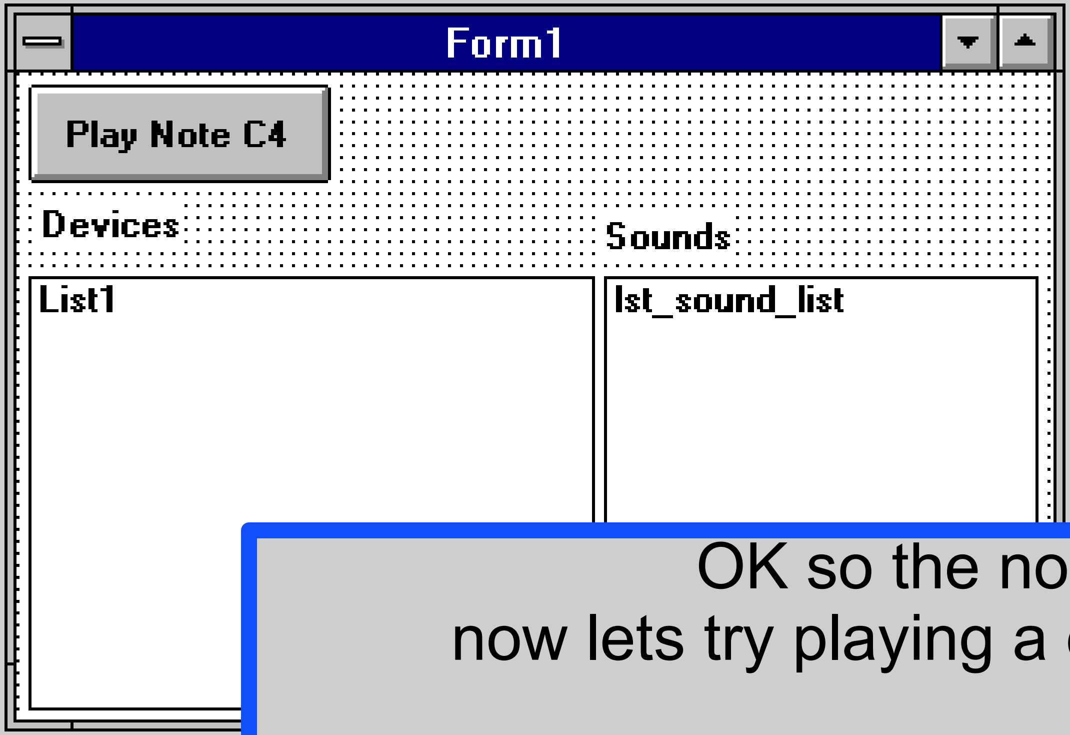


Now we can add a button to test the connection by sending a “note on” MIDI message to the device.



Object: Command1 Proc: Click

```
Sub Command1_Click ()
    Call note_on(0, 60, 127)
End Sub
```

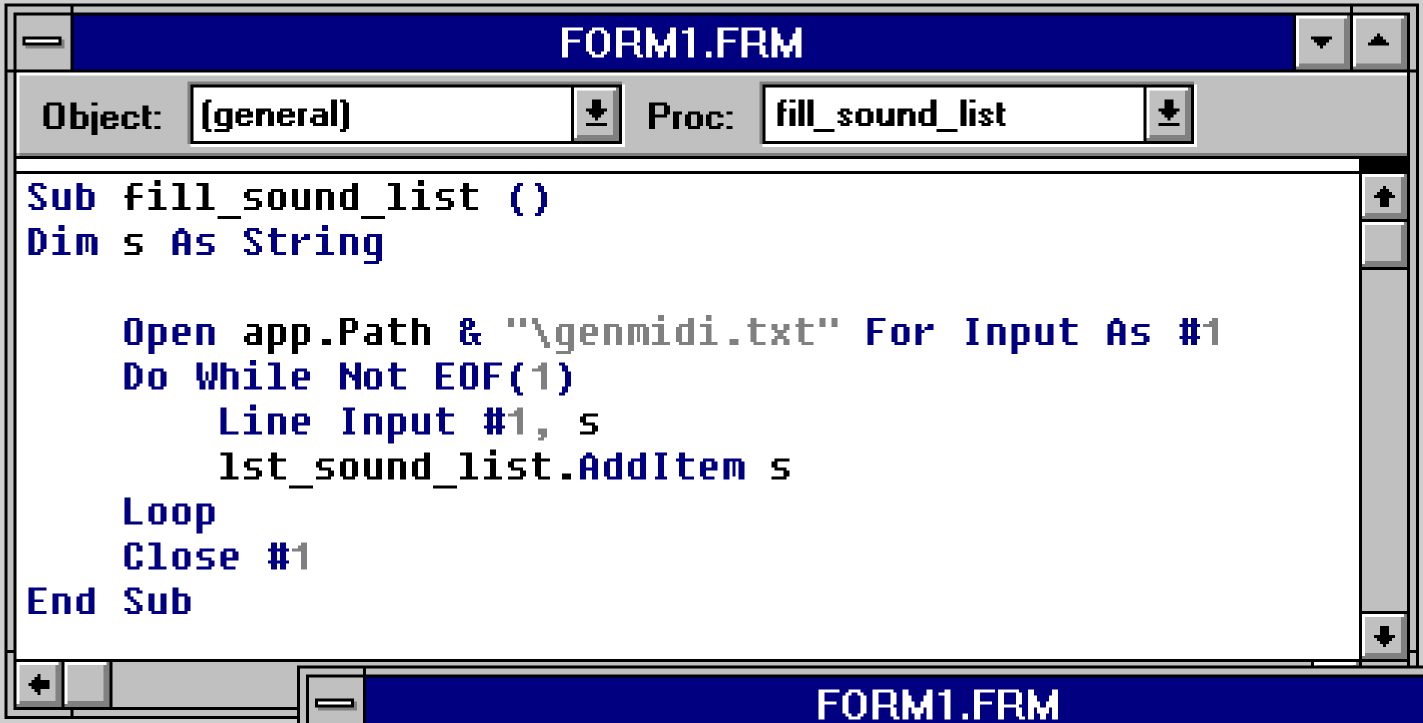


OK so the note sounded
now lets try playing a different instrument.

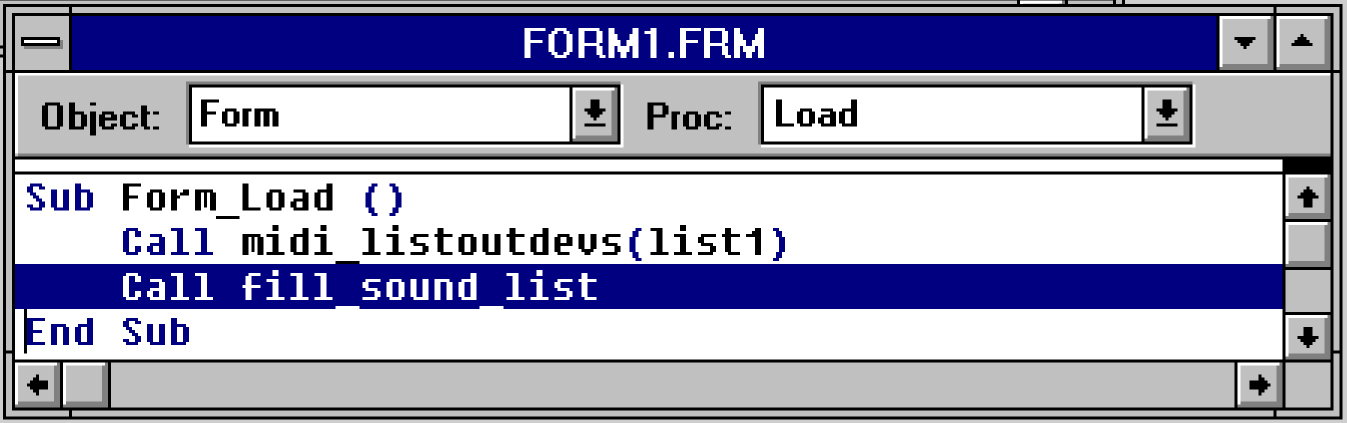
If we load a listbox with instrument names
we can send a MIDI programchange message
each time the user clicks the listbox.

The CD contains a file named genmidi.txt,
in the mmsys1 directory that we can use.

Here is the code to load the listbox.



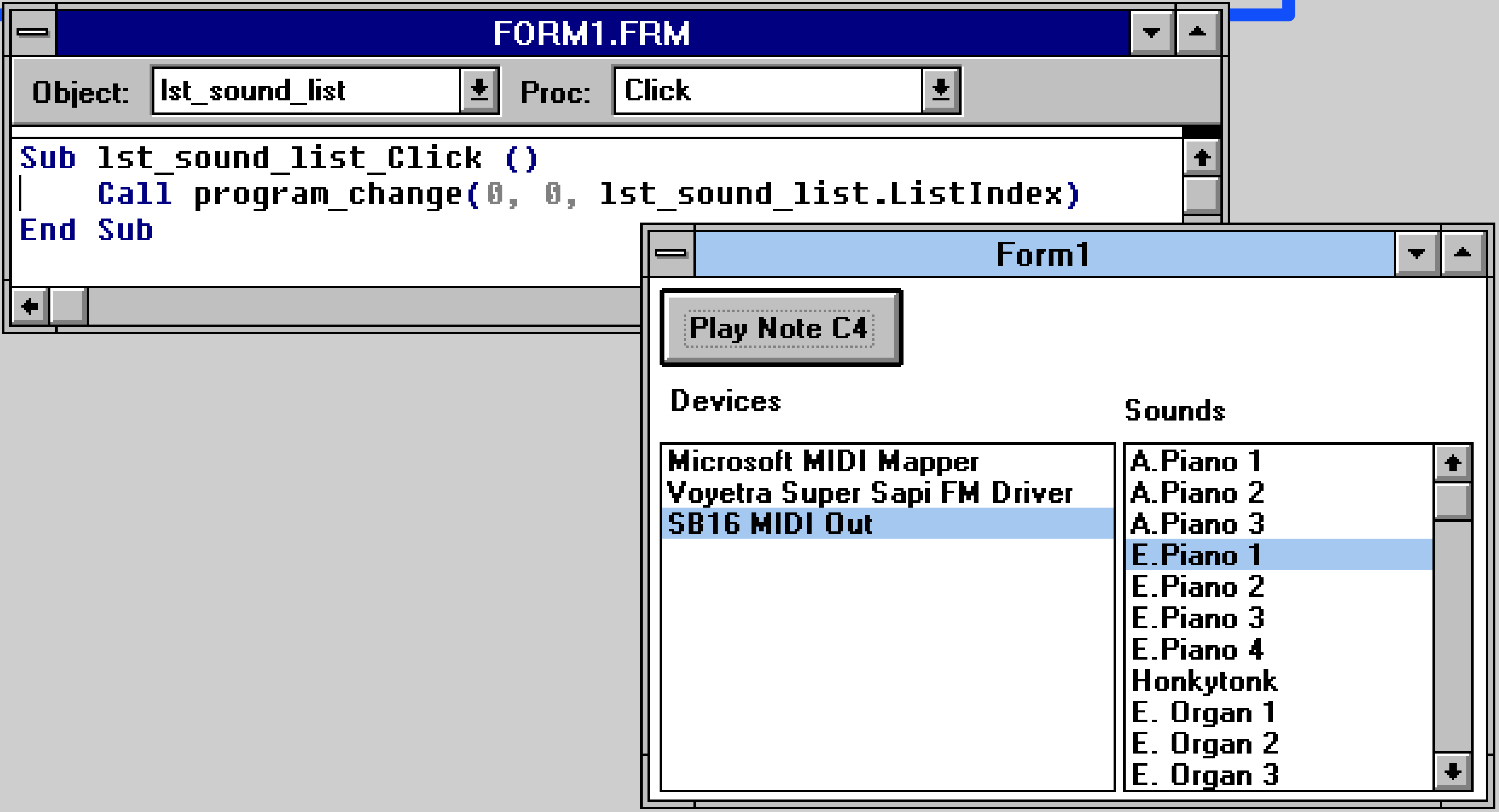
```
Sub fill_sound_list ()
Dim s As String

    Open app.Path & "\genmidi.txt" For Input As #1
    Do While Not EOF(1)
        Line Input #1, s
        lst_sound_list.AddItem s
    Loop
    Close #1
End Sub
```



```
Sub Form_Load ()
    Call midi_listoutdevs(list1)
    Call fill_sound_list
End Sub
```

- and here is the code to send the programchange.

This assumes that the name of the instrument correlates with the position in the list box. (So don't turn on sort...)



FORM1.FRM

Object: lst_sound_list Proc: Click

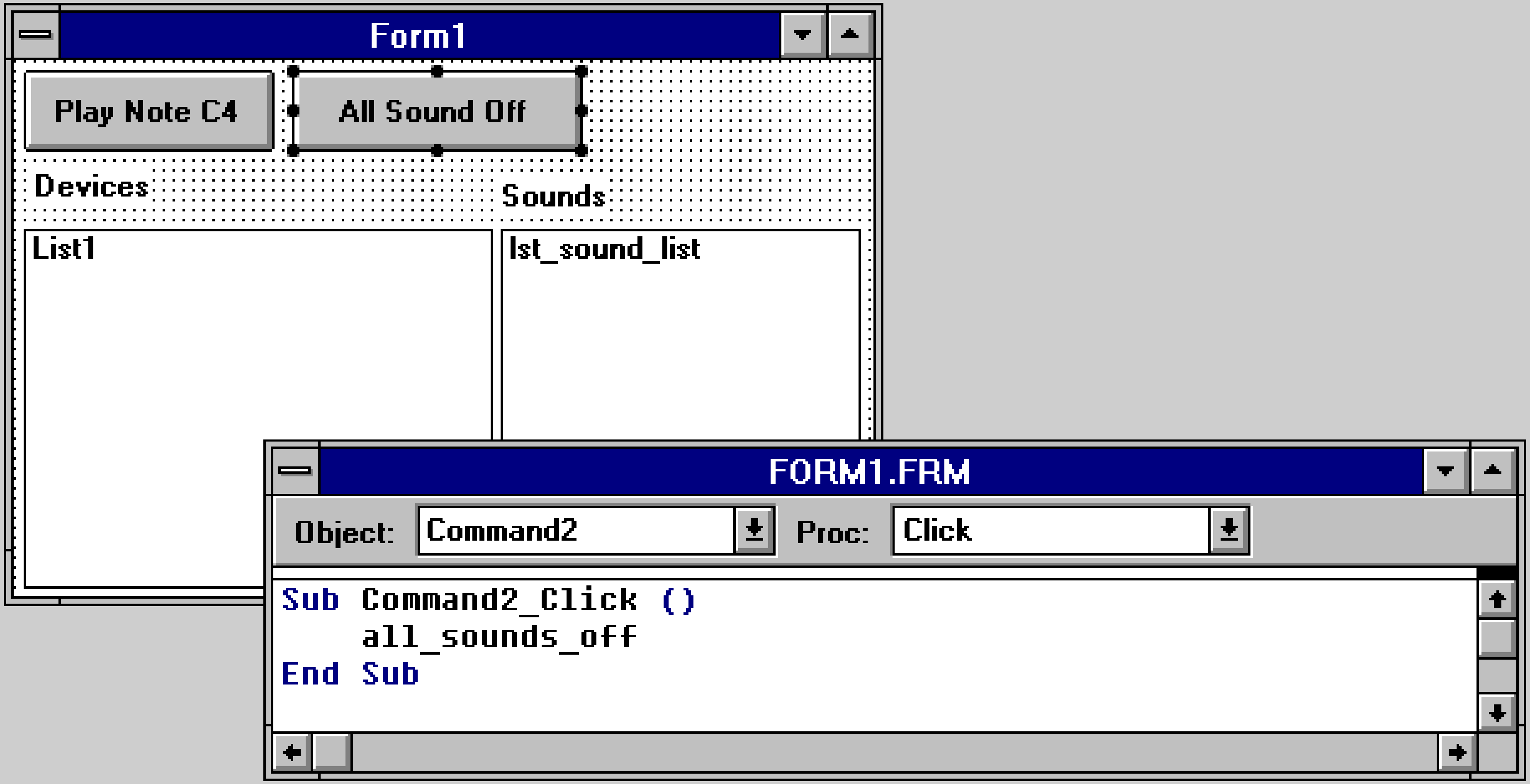
```
Sub lst_sound_list_Click ()
    Call program_change(0, 0, lst_sound_list.ListIndex)
End Sub
```

Form1

Microsoft MIDI Mapper
Voyetra Super Sapi FM Driver
SB16 MIDI Out

A.Piano 1
A.Piano 2
A.Piano 3
E.Piano 1
E.Piano 2
E.Piano 3
E.Piano 4
Honkytonk
E. Organ 1
E. Organ 2
E. Organ 3

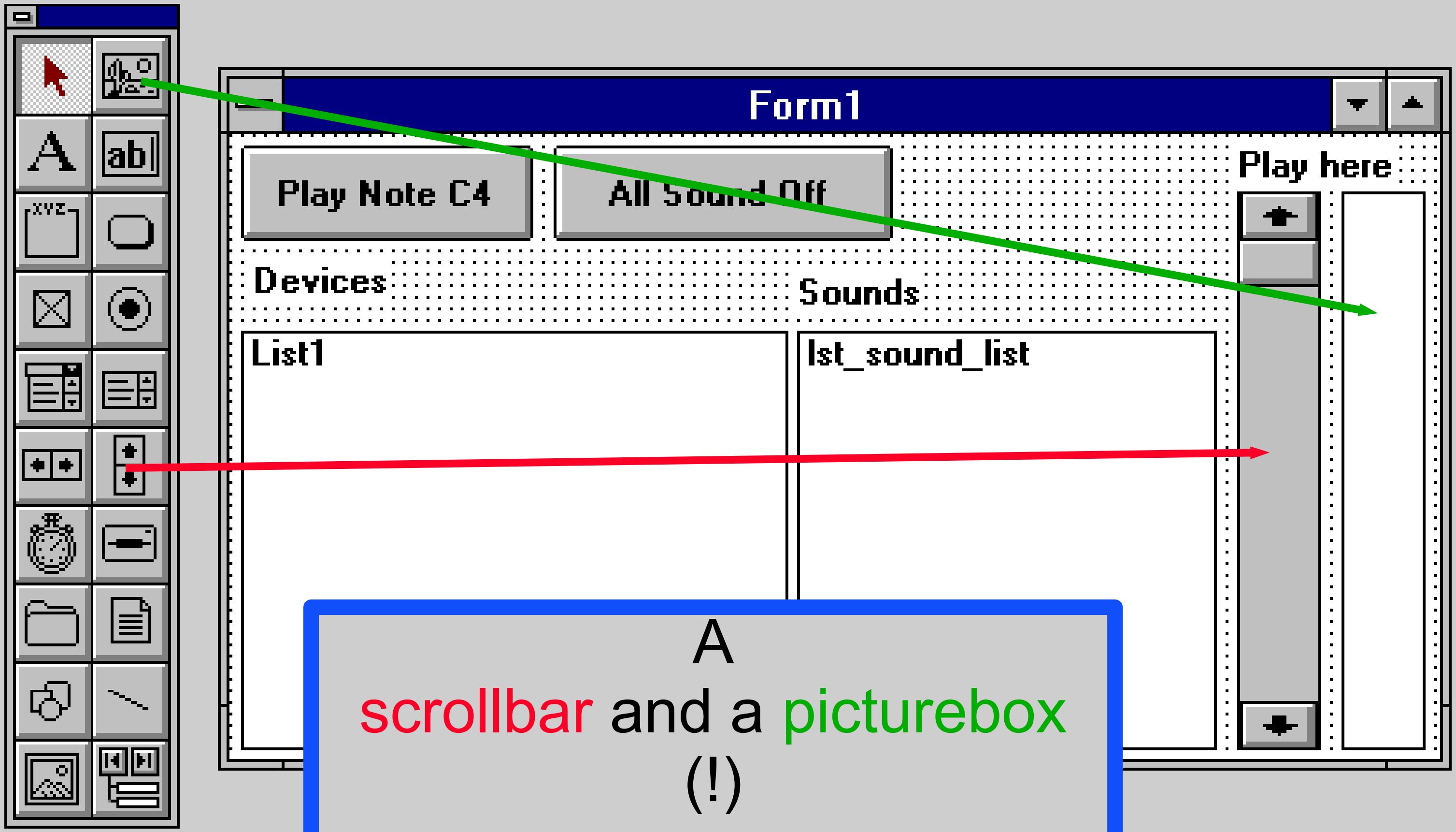
After some testing
(with organs)
you will put on a button to
turn off all sound.....



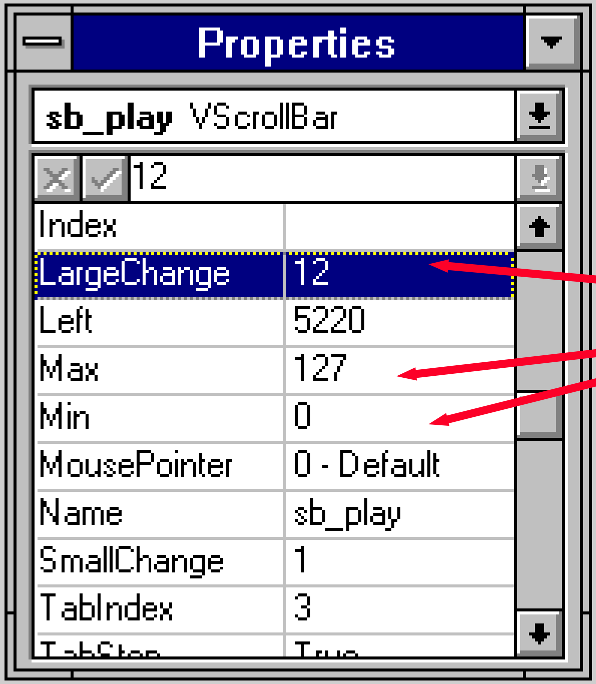
Object: Command2 Proc: Click

```
Sub Command2_Click ()
    all_sounds_off
End Sub
```

Playing C4 all the time gets more that boring so
(if we don't have a keyboard (..music keyboard I meant))
we invent some windows way of playing notes
(Just to test the instrument sounds.....)



A
scrollbar and a picturebox
(!)



Properties

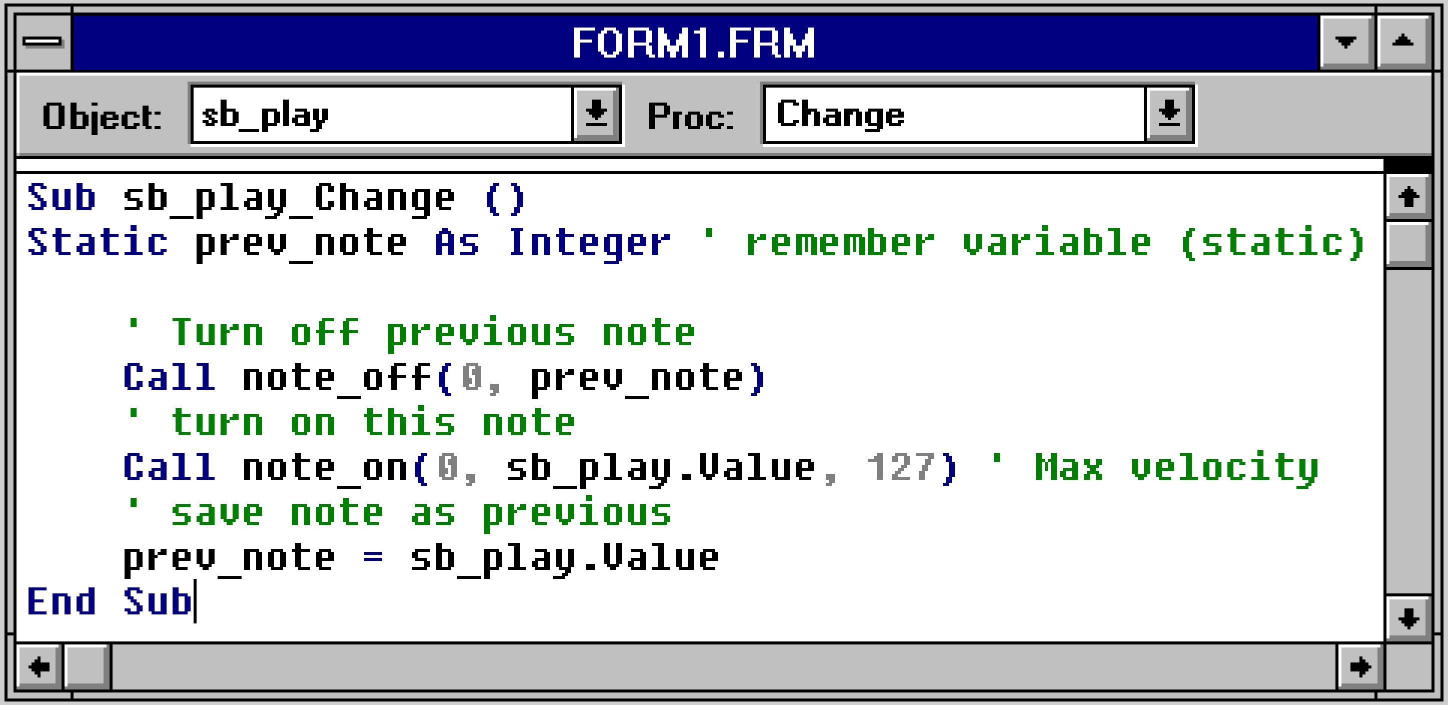
sb_play VScrollBar

12

Index	
LargeChange	12
Left	5220
Max	127
Min	0
MousePointer	0 - Default
Name	sb_play
SmallChange	1
TabIndex	3
TabStop	True

Please note that the max and min properties are set to 127 and 0 (the range of playable notes)

The LargeChange property is set to 12 so that scrolling this way will scroll in octaves.

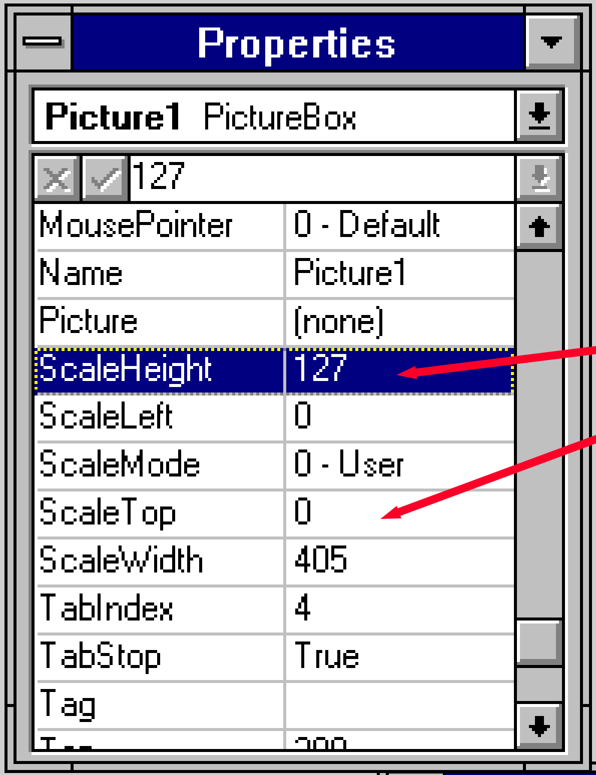


FORM1.FRM

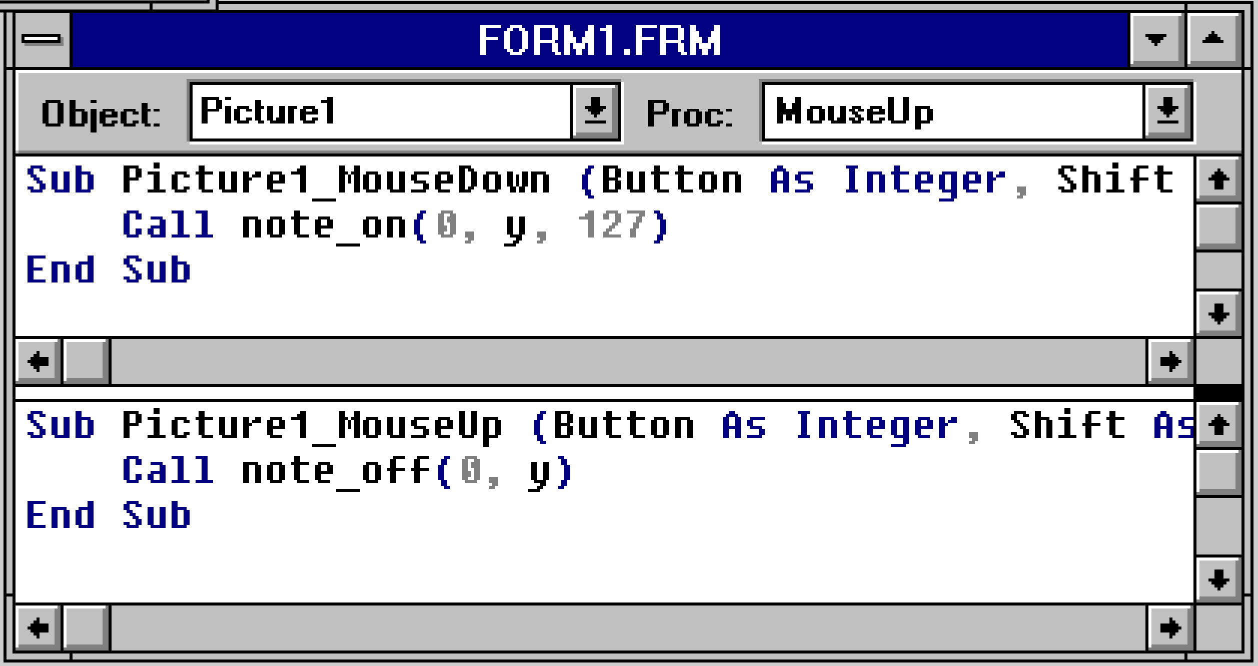
Object: sb_play Proc: Change

```
Sub sb_play_Change ()
Static prev_note As Integer ' remember variable (static)

    ' Turn off previous note
    Call note_off(0, prev_note)
    ' turn on this note
    Call note_on(0, sb_play.Value, 127) ' Max velocity
    ' save note as previous
    prev_note = sb_play.Value
End Sub
```

For the picturebox
the ScaleTop is set to 0
and the ScaleHeight is set to 127
so we can take the mouse y co-ordinate
and send directly to the note on.



MousePointer	0 - Default
Name	Picture1
Picture	(none)
ScaleHeight	127
ScaleLeft	0
ScaleMode	0 - User
ScaleTop	0
ScaleWidth	405
TabIndex	4
TabStop	True
Tag	

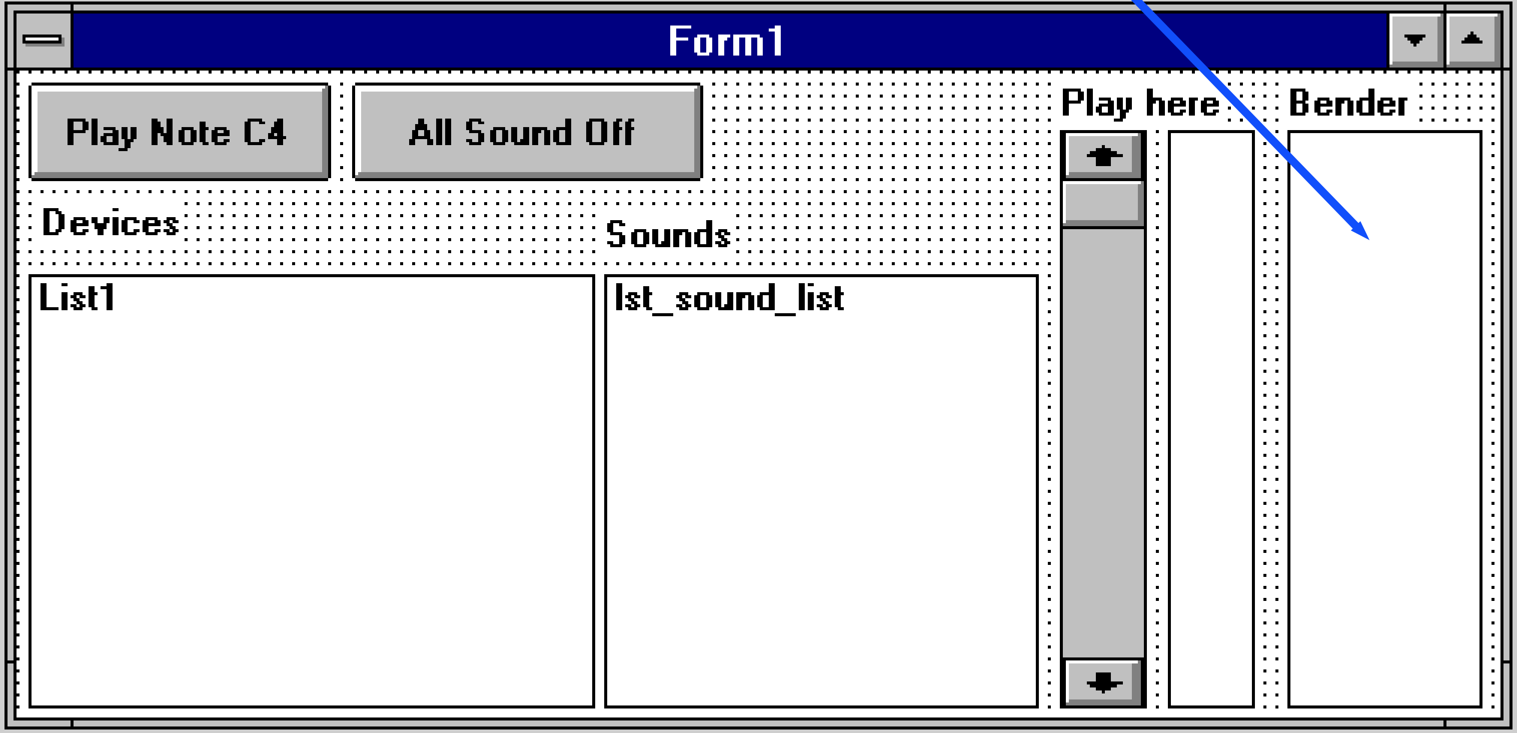


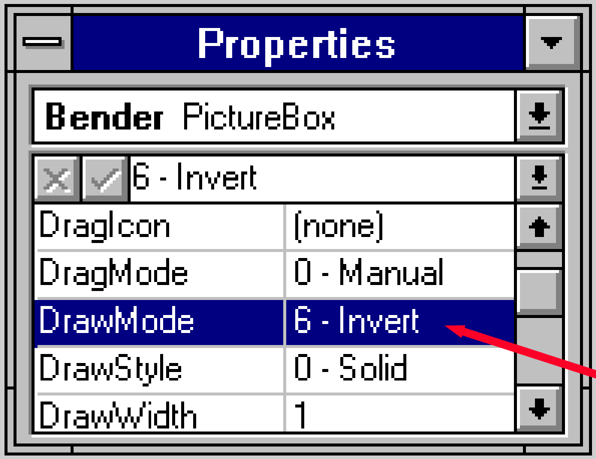
```
Sub Picture1_MouseDown (Button As Integer, Shift
    Call note_on(0, y, 127)
End Sub
```

```
Sub Picture1_MouseUp (Button As Integer, Shift As
    Call note_off(0, y)
End Sub
```

The final kick is to implement a "mouse-bender" from a picturebox !

It will draw a line that follows the mouse when it is inside the box.





DragIcon	(none)
DragMode	0 - Manual
DrawMode	6 - Invert
DrawStyle	0 - Solid
DrawWidth	1

Observe properties

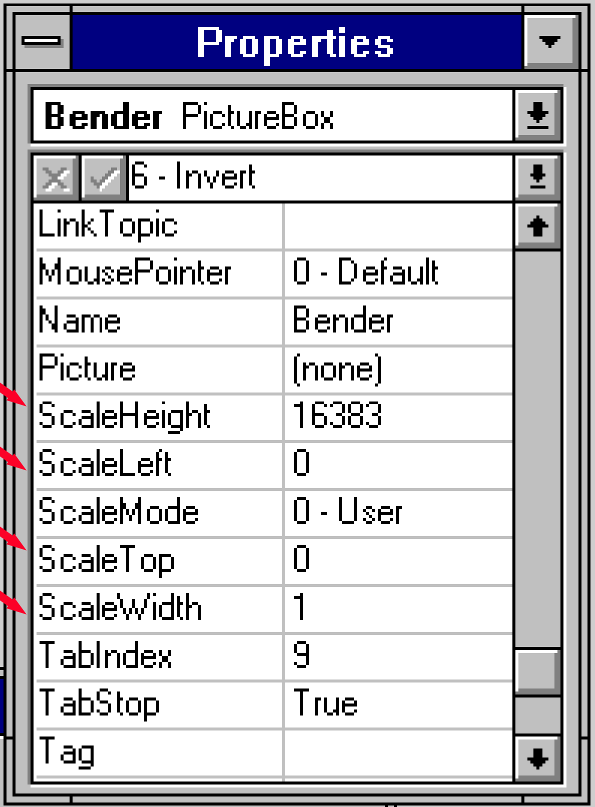
DrawMode

ScaleTop

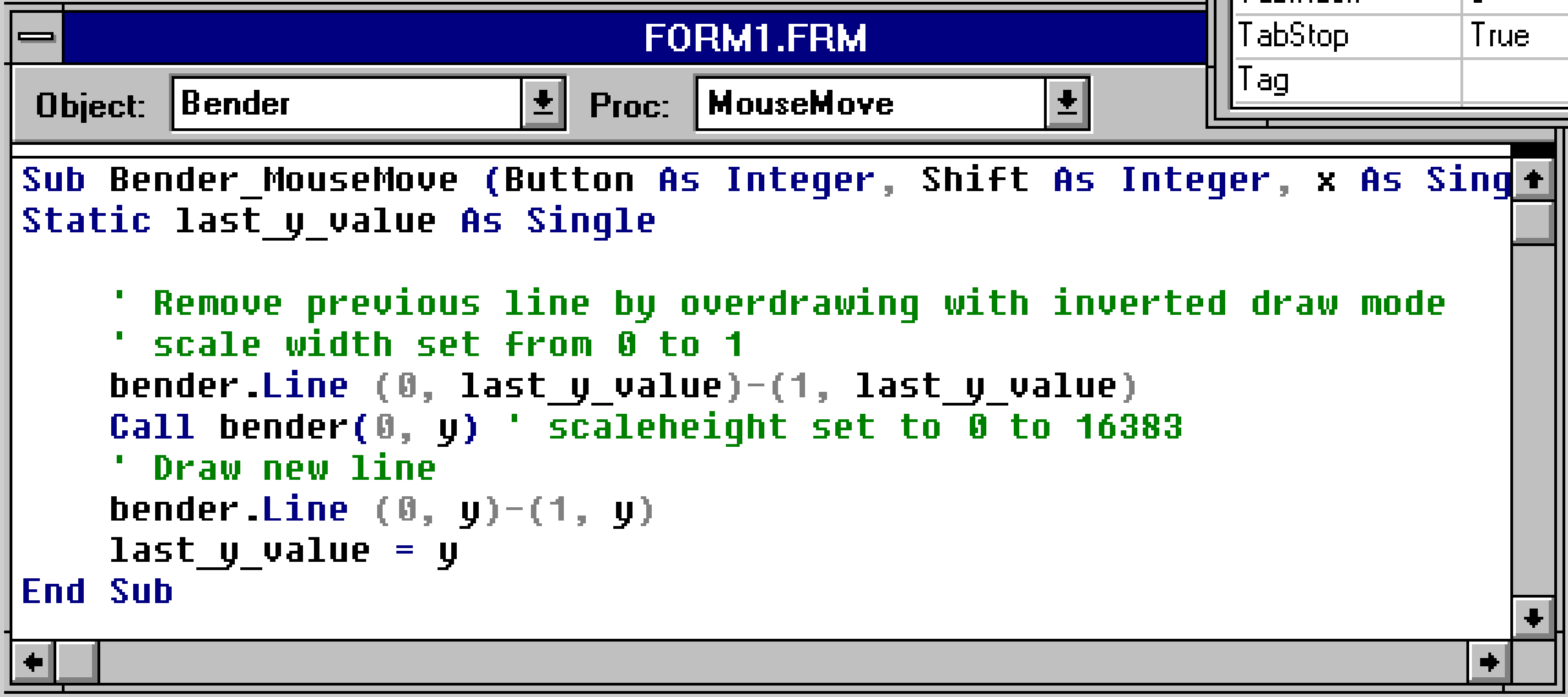
ScaleLeft

ScaleHeight

ScaleWidth



LinkTopic	
MousePointer	0 - Default
Name	Bender
Picture	(none)
ScaleHeight	16383
ScaleLeft	0
ScaleMode	0 - User
ScaleTop	0
ScaleWidth	1
TabIndex	9
TabStop	True
Tag	



```
Sub Bender_MouseMove (Button As Integer, Shift As Integer, x As Sing
Static last_y_value As Single

    ' Remove previous line by overdrawing with inverted draw mode
    ' scale width set from 0 to 1
    bender.Line (0, last_y_value)-(1, last_y_value)
    Call bender(0, y) ' scaleheight set to 0 to 16383
    ' Draw new line
    bender.Line (0, y)-(1, y)
    last_y_value = y
End Sub
```

We have now managed to make a completely useless application that demonstrates a lot of basic MIDI messages and the way to send them

Now you can start building your own application with personal functionality.

Remember to check out the higher level CoolTools.

for this point in the MMSYSTEM\MMSYS1 Directory on this CD-ROM.