Column #94 February 2003 by Jon Williams:

Gettin' MIDI With It

Well, last month I introduced you to the new PBASIC compiler, so now it's time to have some fun and put it to work. Even though we're going to keep things simple, you'll be able to see that PBASIC 2.5 lets you write cleaner code that's even easier to maintain. You'll probably find – as I have – that you can actually write programs a bit faster because there is no more forward-thinking about GOTO labels for IF-THEN. Let's get started.

You'd be hard pressed to find anyone who doesn't like music; in fact, most of us like several varieties. Having experienced the late 60's and early 70's as a youngster, I lean toward what is now called "classic" rock (Led Zeppelin, Jimi Hendricks, etc.), but I also like Classical, Grunge and even a bit of Rap. Music is probably as old as mankind and is unique in that it not only helps define cultures, yet so easily crosses cultural boundaries.

Music and electronics go way back too. Pop quiz: What was the first electronic musical instrument? If you said the Theremin, go get yourself a cookie – you win. The Theremin was invented in 1919 by Russian physicist Lev Termen (who became Leon Theremin). The Theremin became very popular for music and sound effects in horror and science fiction movies, and even found its way into popular music (the Beach Boys used a Theremin-like instrument on "Good Vibrations" and Led Zeppelin guitarist Jimmy Page frequently used a Theremin on stage).

Since I like music (I can even thrash out a couple Counting Crows songs on my guitar) and I love BASIC Stamps, it just makes sense that I should work with both at the same time. And, in December I was out in our Rocklin office and saw a Stamp-controlled xylophone that one of my colleagues, Stephen, had created. That same week, I assisted a New York professor who has the interesting occupation of teaching music and technology. Music AND technology … that sounds like a lot of fun.

Musical Connections

While the Theremin was the first electronic musical instrument, it was (and still is) a specialty device and never became mainstream. The electronic synthesizer, however, is a different story altogether. In the early 1980's a group of synthesizer manufacturers got together to create a standard that would allow various instruments to connect to and control each other. The standard they created is called MIDI: Musical Instrument Digital Interface.

The MIDI specification (see www.midi.org) actually has three components: the protocol, the connection and a file format. For our part, we're going to focus on the protocol. Of course, we'll also take a look at the connection since our goal is to have the BASIC Stamp play music on a MIDI-compatible instrument.

The MIDI protocol, as it turns out, is very straightforward and easy to implement; even on a small micro like the BASIC Stamp. The protocol transmission scheme is straight serial at 31.25 kBaud; a value we're not generally used to but not a problem for the BASIC Stamp since we can calculate and set the baudmode parameter of SEROUT. For the BS2, the calculation of the baudmode parameter is:

INT ((1,000,000 / baud) – 20)

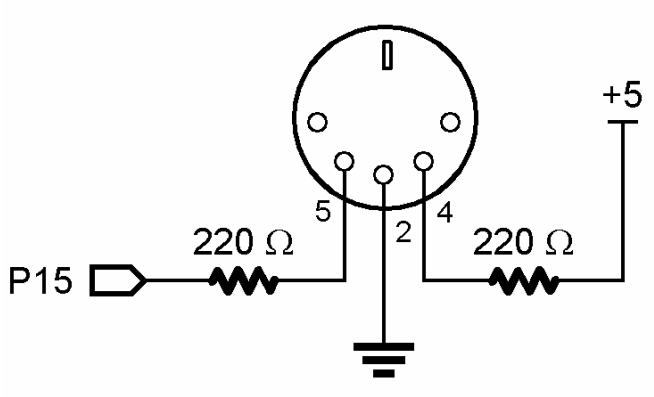
For MIDI we get:

(1,000,000 / 31,250) – 20 = 12

Since the MIDI interface standard uses an optical-isolator on the input that is pulled up by the connecting device, we can run in "open" mode. To do this, we'll add $8000 to the baudmode.

Figure 94.1 shows the connection to the BASIC Stamp. As you can see, the hardware is a cinch: a couple 220-ohm resistors and a 5-pin DIN socket (female). This interface will let you connect your BASIC Stamp to MIDI instrument using an off-the-shelf MIDI cable (you can pick one up at Radio Shack®).

Figure 94.1: BASIC Stamp to MIDI Connector



MIDI Messages

MIDI message packets are small; usually a command followed by two data bytes. Since our goal this month is to allow the BASIC Stamp to play music through a MIDI instrument, the two messages we are going to be concerned most with are "Note On" ($90) and "Note Off" ($80).

Here's the syntax:

Note On:	$90, note (0 – 127), velocity (0 – 127)
Note Off:	$80, note (0 – 127), velocity (0 – 127)

If, for example, we want to send a "Note On" for Middle C at the loudest volume, we would do this:

```
SEROUT 15, $8000+12, [$90, $3C, $7F]
```

Pretty simple, isn't it? Yes, it is. Let's go ahead and create a program to experiment with sending MIDI messages. Of course, you'll need an instrument to play through. I used a MIDI-compatible keyboard (that, until I started this project, was the most expensive guitar tuner I own…).

Our program should let us specify a number of bytes to send (1 to 3), then accept the data and transmit it as a packet when we're ready. If you're new to BASIC Stamps, this may sound a bit complicated but, as you'll see, this is actually a very simple application.

The first part of our program simply clears the DEBUG window and displays the title. Notice that there is a short PAUSE before the title display. This momentary blank screen serves as a visual cue. We'll get into that more in just a second.

```
Main:
  DO
    DEBUG CLS
    PAUSE 500
    DEBUG "MIDI Explorer", CR
    DEBUG "-------------"
```

As I may have mentioned last month, the new compiler supports some new named constants; many that are used specifically with DEBUG. The first that we'll use is CrsrXY which positions the DEBUG cursor at the column and line values that follow. The next is ClrDn. This causes the DEBUG window to clear from the cursor line down .

After clearing any old input, we prompt ourselves for the number of bytes to send, then accept that value from the DEBUG window (SERIN on pin 16 at 9600 baud).

```
    DEBUG CrsrXY, 0, 3, ClrDn
    DEBUG "Bytes to send? : "
    SERIN 16, 84, [DEC1 nBytes]
```

The following section does most of the work. First we'll test for a legal nBytes value. If the value is okay, we'll loop through the number of bytes and take each one in as a two-digit hex value. Note the use of CrsrXY again and accepting serial data from the DEBUG window. If the nBytes value is out-of-range, the THEN portion of IF-THEN won't run and the code will LOOP back to the beginning (momentary blank screen).

```
    IF (nBytes > 0) AND (nBytes <= 3) THEN
      FOR idx = 1 TO nBytes
        DEBUG CrsrXY, 0, (idx + 4)
        DEBUG "(", DEC1 idx, ") : "
        SERIN 16, 84, [HEX2 midi(idx - 1)]
      NEXT
```

With the packet in memory, we'll WAIT on the Enter (CR) key to be pressed before sending it.

```
      DEBUG CrsrXY, 0, (nBytes + 6)
      DEBUG "Press [Enter] to execute. "
      SERIN 16, 84, [WAIT (CR)]
```

The last step, then, is to send the data. The STR modifier and \Length parameter make it easy to send the MIDI data. This works particularly well for us since we may want to change the number of bytes to send.

```
      SEROUT MidiOut, MidiBaud, [STR midi\nBytes]
    ENDIF
  LOOP

  END
```

Figure 94.2 shows the program in action. Something interesting that you may find with your instrument that I found with mine is that two consecutive Note On commands for the same note require two Note Off commands. This becomes very apparent with voice selections that don't have natural decay (volume fade).

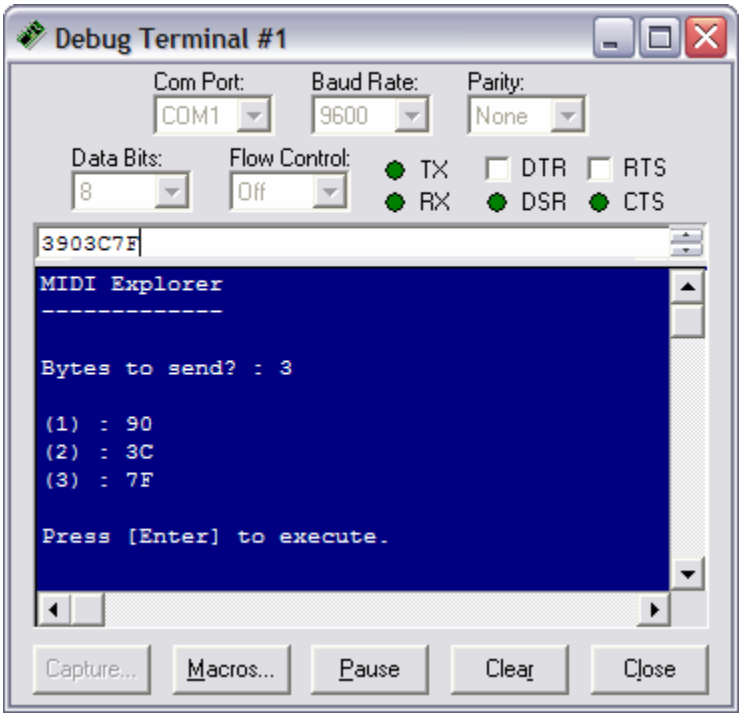
Okay, if you were able to turn a note on and off, then you can actually play music via your BASIC Stamp. Why would you want to do this? Well, there's a lot of reasons including the fact that there is hardware available that responds to MIDI control signals (lighting, for example).

Stamp-Made Music

If we can send one note at a time, we can send a whole bunch – in a specific order – to play music, right? You betcha.

Like the first, our second program is fairly short [the working part], but this one is just a bit more sophisticated. The purpose of this program is to play music stored in BASIC Stamp DATA statements.

Figure 94.1: DEBUG Window "MIDI Explorer"



The DATA statement is one of those features that migrated from "classic" PC BASIC implementations, though the Stamp makes it easier and more functional. Remember that the BASIC Stamp compiler writes the program's DATA statements to EEPROM, beginning at address 0 (unless specified otherwise) and building up. Program tokens are stored in the top of the EEPROM and build down. If there is a class between DATA and program tokens, the compiler will let you know.

There are several key features of the BASIC Stamp DATA statement that make it more flexible that past PC implementations. The feature I take most advantage of is the ability to name DATA locations. Look at this code:

```
Name           DATA    "Jon Williams", 0
Location       DATA    "Dallas, TX  USA", 0
```

What we've done here is defined two zero-terminated strings. To print a string in the DEBUG window we can use this subroutine:

```
Print_String:
  READ eePntr, char
  IF (char <> 0) THEN
    DEBUG char
    eePntr = eePntr + 1
    GOTO Print_String
  ENDIF
  RETURN
```

The subroutine needs to know where to get characters – the address of the string is put into the variable eePntr. So, if we do this:

```
  eePntr = Name
  GOSUB Print_String
```

…what's actually happening is that the compiler is evaluating the DATA statement labels into constant values that represent the EEPROM location of the first character in each string. Given no other DATA statements, Name evaluates to zero and Location evaluates to 13.

Another neat thing about BASIC Stamp DATA statements is the ability to store any kind of information that we want. Here's another example:

```
Name           DATA    "Jon Williams", 0
Birthday       DATA    7, 25
YrOfBirth      DATA    Word 1962
```

As you can see we've got a string, two Bytes and a Word (and yes, that's really my birthday – don't forget the cards and gifts!). Actually, all data is store as Bytes. For character and string data, the ASCII value is stored. For Words, they're stored Little-Endian, that is, low-byte first. As I pointed out last month, PBASIC 2.5 allows us to READ a Word with just one operation.

Finally, the information in our DATA statements is stored in EEPROM, which means we can change it at run-time with WRITE. This allows our program to change information and maintain it even when power is lost.

Sing Me A Song, You're The Stamp-Man

Okay, let's play some music. The reason I spent a moment discussing the details of the DATA statement is that's where where' going to keep our song information. The key to this program is how we're going to store the song information.

While experimenting I found that I could actually cause more than one note to play with the same "Note On" command. I simply stacked the note and velocity bytes behind it. Like this:

$90, note1, velocity1, note2, velocity2, note3, velocity3

By playing three notes at a time, I could play simple chords and make something a bit more musical. Okay! In the end, I came up with this storage strategy:

command, notes, timing, note1, velocity1 {, note2, velocity2, note3, velocity3}

Note that the items in curly braces are optional. Here's an actual line from the program's DATA table to illustrate:

```
  DATA    NN, 3, Word N01, 060, 100, 064, 100, 067, 100
```

A few constants are used here. NN is the constant value for Note On ($90). N01 is the constant value for a whole-note duration. This line of DATA represents a simple chord made up of the notes C4 (Middle C), E4 and G4. All three notes are played at a velocity (volume) of 100.

All right, it's time to look at the program in action. Here's the main program code:

```
Main:
  eePntr = Mystery
  GOSUB Play_Song
  END
```

Really, that's the program – clearly all the work is done in Play_Song and our previous discussion explains the use of eePntr and how it indicates one song from many stored in memory. Let's have a look at Play_Song, then we'll go through it step-by-step.

```
Play_Song:
  READ eePntr, cmd
  IF (cmd < $FF) THEN
    READ (eePntr + 1), notes
    READ (eePntr + 2), Word nTime
    FOR idx = 0 TO (notes * 2 - 1)
      READ (eePntr + 4 + idx), midi(idx)
    NEXT
    SEROUT MidiOut, MidiBaud, [cmd, STR midi\(notes * 2)]
    PAUSE nTime
    eePntr = eePntr + 6 + ((notes - 1) * 2)
    GOTO Play_Song
  ENDIF
  RETURN
```

This code is actually quite simple, even if it doesn't appear that way at first glance. We start by reading the command byte. The program will use $FF as an end-of-song indicator, so we have to test for that first. If the command byte is not $FF, the rest of the subroutine will run, otherwise we RETURN to the caller.

Assuming we have a valid command, next up is the number of notes (1 to 3) to play. After the notes is the timing for them. As you can see, we're taking advantage of the new Word modifier for READ. The timing value represents the last piece of fixed information. The number of bytes (for notes and velocities) varies; we could have two bytes, four bytes or six bytes.

A FOR-NEXT loop is used to iterate through to the end of this line and store the notes and velocities in an array called midi. All that's left to do now is to send the command and data to our MIDI device. Once that's taken care of, we'll use PAUSE to create the note duration. We've got music. The last step is to update eePntr and go back to the top of Play_Song.

To be honest, the trickiest part of this whole process was converting music into the DATA statements. If, for example, we needed to play a C4 and a D4 at the same time, with the C4 being a quarter-note long and the D4 being a half-note long, the DATA looks like this:

```
  DATA    NN, 2, Word N04, 060, 100, 062, 100
  DATA    NX, 1, Word N04, 060, 000
  DATA    NX, 1, Word 000, 062, 000
```

Can you see what's happening? The first line strikes both notes and holds them for a quarter-note duration. The second line stops the C4 and holds for an additional quarter-note duration. Remember, the D4 is still playing, so it is now a half-note long. Finally, the D4 is stopped.

I used a demo version of Sonar (from www.cakewalk.com) to read a MIDI file and display it in standard musical notation. Thankfully, Sonar has this really neat feature that displays note properties and velocity. The notes are shown as C4, G6, etc. To convert to numeric values, I used a table found at this link:

www.harmony-central.com/MIDI/Doc/table2.html

I'll tell you what … the first person to identify the song in the main program will win a BASIC Stamp – a brand new BS2pe-24. How about that? What that means is you have to download the code, build the interface and let it play. Send me an e-mail with your answer.

Until next time, Happy Valentines day and Happy Stamping.

```
' =============================================================================
'
'   File...... Midi Explorer.BS2
'   Purpose... Sends midi packets to instrument
'   Author.... Jon Williams, Parallax
'   E-mail.... jwilliams@parallax.com
'   Started...
'   Updated... 22 DEC 2002
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------
'
' This program allows the user to enter and send a midi packet through
' the DEBUG terminal.  It is designed for exploring midi byte values and
' their behavior.


' -----[ Revision History ]------------------------------------------------


' -----[ I/O Definitions ]-------------------------------------------------

MidiOut         PIN     15                      ' midi serial output


' -----[ Constants ]-------------------------------------------------------

MidiBaud        CON     $8000 + 12              ' 31.25 kBaud -- open


' -----[ Variables ]-------------------------------------------------------

nBytes          VAR     Nib                     ' number of bytes to send
idx             VAR     Nib
midi            VAR     Byte(3)


' -----[ EEPROM Data ]-----------------------------------------------------


' -----[ Initialization ]--------------------------------------------------


' -----[ Program Code ]----------------------------------------------------
```

```
Main:
  DO
    DEBUG CLS
    PAUSE 500
    DEBUG "MIDI Explorer", CR
    DEBUG "-------------"

    DEBUG CrsrXY, 0, 3, ClrDn                  ' clear old input
    DEBUG "Bytes to send? : "                  ' prompt for nBytes
    SERIN 16, 84, [DEC1 nBytes]                ' get nBytes from user
    IF (nBytes > 0) AND (nBytes <= 3) THEN     ' test nByte value
      FOR idx = 1 TO nBytes
        DEBUG CrsrXY, 0, (idx + 4)             ' move to input line
        DEBUG "(", DEC1 idx, ") : "            ' prompt byte input
        SERIN 16, 84, [HEX2 midi(idx - 1)]     ' get midi byte
      NEXT
      DEBUG CrsrXY, 0, (nBytes + 6)
      DEBUG "Press [Enter] to execute. "
      SERIN 16, 84, [WAIT (CR)]                ' wait for [Enter]
      SEROUT MidiOut, MidiBaud, [STR midi\nBytes]
    ENDIF
  LOOP

  END


' -----[ Subroutines ]------------------------------------------------------
```

```
' =============================================================================
'
'   File...... Stamp Midi Player.BS2
'   Purpose... Midi demo with BASIC Stamp 2
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallax.com
'   Started... 22 DEC 2002
'   Updated... 24 DEC 2002
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------
'
' This program demonstrates a very simple mechanism for storing songs in
' EEPROM and playing them through a connected midi instrument.  The song
' is played to the channel 1 of the instrument.


' -----[ Revision History ]------------------------------------------------


' -----[ I/O Definitions ]-------------------------------------------------

MidiOut         PIN     15                      ' midi serial output


' -----[ Constants ]-------------------------------------------------------

MidiBaud        CON     $8000 + 12              ' 31.25 kBaud -- open

Channel         CON     0
NN              CON     $90 | Channel           ' note on
NX              CON     $80 | Channel           ' note off

N01             CON     1600                    ' whole note
ND2             CON     N01 / 4 * 3             ' dotted half
N02             CON     N01 / 2                 ' half note
N04             CON     N01 / 4                 ' quarter note
N08             CON     N01 / 8                 ' eighth note
N12             CON     N04 / 3                 ' quarter note triplet
N16             CON     N01 / 16
N32             CON     N01 / 32


' -----[ Variables ]-------------------------------------------------------

eePntr          VAR     Word                    ' pointer to EE table
```

```
cmd             VAR     Byte                    ' command (on or off)
notes           VAR     Nib                     ' number of notes to play
nTime           VAR     Word                    ' note timing
midi            VAR     Byte(6)                 ' up to 3 notes at once

idx             VAR     Nib


' -----[ EEPROM Data ]-------------------------------------------------

' Record format
' Command, Notes, Timing, Note1, Volume1 {, Note2, Volume2, Note3, Volume3}

Mystery         DATA    NN, 1, Word N12, 069, 100
                DATA    NX, 1, Word 000, 069, 000
                DATA    NN, 1, Word N12, 072, 060
                DATA    NX, 1, Word 000, 072, 000
                DATA    NN, 1, Word N12, 076, 060
                DATA    NX, 1, Word 000, 076, 000
                DATA    NN, 1, Word N12, 077, 072
                DATA    NX, 1, Word 000, 077, 072
                DATA    NN, 1, Word N12, 072, 060
                DATA    NX, 1, Word 000, 072, 000
                DATA    NN, 1, Word N12, 076, 060
                DATA    NX, 1, Word 000, 076, 000
                DATA    NN, 1, Word N12, 077, 072
                DATA    NX, 1, Word 000, 077, 072
                DATA    NN, 1, Word N12, 072, 060
                DATA    NX, 1, Word 000, 072, 000
                DATA    NN, 1, Word N12, 076, 060
                DATA    NX, 1, Word 000, 076, 000
                DATA    NN, 1, Word N04, 083, 074
                DATA    NX, 1, Word 000, 083, 000
                DATA    NN, 1, Word N04, 083, 074
                DATA    NX, 1, Word 000, 083, 000
                DATA    NN, 2, Word N04, 083, 074, 081, 080
                DATA    NX, 2, Word 000, 083, 000, 081, 000
                DATA    NN, 2, Word N04, 083, 074, 088, 080
                DATA    NX, 2, Word 000, 083, 000, 088, 000
                DATA    NN, 2, Word N12, 069, 100, 086, 100
                DATA    NX, 1, Word 000, 069, 000
                DATA    NN, 1, Word N12, 072, 060
                DATA    NX, 1, Word 000, 072, 000
                DATA    NN, 1, Word N12, 076, 060
                DATA    NX, 2, Word 000, 076, 000, 086, 000
                DATA    NN, 2, Word N12, 077, 072, 088, 100
                DATA    NX, 1, Word 000, 077, 000
                DATA    NN, 1, Word N12, 072, 060
                DATA    NX, 1, Word 000, 072, 000
                DATA    NN, 1, Word N12, 076, 060
                DATA    NX, 2, Word 000, 076, 000, 088, 000
```

```
                DATA    NN, 2, Word N12, 077, 060, 091, 100
                DATA    NX, 1, Word 000, 077, 000
                DATA    NN, 1, Word N12, 072, 060
                DATA    NX, 1, Word 000, 072, 000
                DATA    NN, 1, Word N12, 076, 060
                DATA    NX, 2, Word 000, 076, 000, 091, 000
                DATA    NN, 2, Word N12, 077, 060, 088, 100
                DATA    NX, 1, Word 000, 077, 000
                DATA    NN, 1, Word N12, 072, 060
                DATA    NX, 1, Word 000, 072, 000
                DATA    NN, 1, Word N12, 076, 060
                DATA    NX, 1, Word ND2, 076, 000
                DATA    NX, 1, Word 000, 088, 000
                DATA    $FF

Chords          DATA    NN, 3, Word N01, 060, 100, 064, 100, 067, 100
                DATA    NX, 3, Word 000, 060, 000, 064, 000, 067, 000
                DATA    NN, 3, Word N01, 060, 100, 063, 100, 067, 100
                DATA    NX, 3, Word 000, 060, 000, 063, 000, 067, 000
                DATA    NN, 3, Word N01, 072, 100, 076, 100, 079, 100
                DATA    NX, 1, Word N01, 000, 000
                DATA    NX, 3, Word 000, 072, 000, 076, 000, 079, 000
                DATA    $FF


' -----[ Initialization ]----------------------------------------------------


' -----[ Program Code ]-------------------------------------------------------

Main:
  eePntr = Mystery
  GOSUB Play_Song
  END


' -----[ Subroutines ]--------------------------------------------------------

Play_Song:
  READ eePntr, cmd                              ' get command
  IF (cmd < $FF) THEN                           ' if valid, play on
    READ (eePntr + 1), notes                    ' get number of notes
    READ (eePntr + 2), Word nTime               ' get note timing
    FOR idx = 0 TO (notes * 2 - 1)              ' read notes/velocities
      READ (eePntr + 4 + idx), midi(idx)
    NEXT
    SEROUT MidiOut, MidiBaud, [cmd, STR midi\(notes * 2)]
    PAUSE nTime                                 ' wait
    eePntr = eePntr + 6 + ((notes - 1) * 2)     ' point to next record
    GOTO Play_Song                              ' keep going
  ENDIF
```

```
RETURN
```

```
' =============================================================================
'
'   File...... Stamp Midi Player (v2).BS2
'   Purpose... Midi demo with BASIC Stamp 2
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallax.com
'   Started... 22 DEC 2002
'   Updated... 24 DEC 2002
'
'   {$STAMP BS2}
'   {$PBASIC 2.5}
'
' =============================================================================


' -----[ Program Description ]---------------------------------------------
'
' This program demonstrates a very simple mechanism for storing songs in
' EEPROM and playing them through a connected midi instrument.  This ver-
' sion lets each note be sent to a different channel (voice).


' -----[ Revision History ]------------------------------------------------


' -----[ I/O Definitions ]-------------------------------------------------

MidiOut         PIN     15                      ' midi serial output


' -----[ Constants ]------------------------------------------------------

MidiBaud        CON     $8000 + 12              ' 31.25 kBaud -- open

NN              CON     $90                     ' note on
NX              CON     $80                     ' note off
CP              CON     $C0                     ' change patch

ND              CON     $01                     ' note duration

N01             CON     1800                    ' whole note
ND2             CON     N01 / 4 * 3             ' dotted half
N02             CON     N01 / 2                 ' half note
N04             CON     N01 / 4                 ' quarter note
N08             CON     N01 / 8                 ' eighth note
N12             CON     N04 / 3                 ' quarter note triplet
N16             CON     N01 / 16
N32             CON     N01 / 32


' -----[ Variables ]------------------------------------------------------
```

```
eePntr          VAR     Word                    ' pointer to EE table
cmd             VAR     Byte                    ' command (on or off)
patch           VAR     Byte                    ' patch (voice)
notes           VAR     Nib                     ' number of notes to play
nTime           VAR     Word                    ' note timing
midi            VAR     Byte(6)                 ' up to 3 notes at once
chan            VAR     Nib                     ' channel

idx             VAR     Nib


' -----[ EEPROM Data ]----------------------------------------------------

Mystery         DATA    CP+0, $3B               ' french horn
                DATA    CP+1, $32               ' guitar
                DATA    CP+2, $1E               ' whistle

                DATA    NN+0, 1, 069, 100, ND, Word N12
                DATA    NX+0, 1, 069, 000
                DATA    NN+0, 1, 072, 060, ND, Word N12
                DATA    NX+0, 1, 072, 000
                DATA    NN+0, 1, 069, 060, ND, Word N12
                DATA    NX+0, 1, 069, 000
                DATA    NN+0, 1, 077, 072, ND, Word N12
                DATA    NX+0, 1, 077, 000
                DATA    NN+0, 1, 072, 060, ND, Word N12
                DATA    NX+0, 1, 072, 000
                DATA    NN+0, 1, 069, 060, ND, Word N12
                DATA    NX+0, 1, 069, 000
                DATA    NN+0, 1, 077, 072, ND, Word N12
                DATA    NX+0, 1, 077, 000
                DATA    NN+0, 1, 072, 060, ND, Word N12
                DATA    NX+0, 1, 072, 000
                DATA    NN+0, 1, 069, 060, ND, Word N12
                DATA    NX+0, 1, 069, 000
                DATA    NN+0, 1, 077, 072, ND, Word N12
                DATA    NX+0, 1, 077, 000
                DATA    NN+0, 1, 072, 060, ND, Word N12
                DATA    NX+0, 1, 072, 000
                DATA    NN+0, 1, 069, 060, ND, Word N12
                DATA    NX+0, 1, 069, 000

                DATA    NN+1, 1, 081, 074, ND, Word N04
                DATA    NX+1, 1, 081, 000
                DATA    NN+1, 1, 081, 067, ND, Word N04
                DATA    NX+1, 1, 081, 000
                DATA    NN+1, 1, 081, 066, NN+2, 1, 081, 079, ND, Word N04
                DATA    NX+1, 1, 081, 000, NX+2, 1, 081, 000
                DATA    NN+1, 1, 081, 038, NN+2, 1, 088, 072, ND, Word N04
                DATA    NX+1, 1, 081, 000, NX+2, 1, 088, 000
```

```
                DATA    NN+0, 1, 069, 100, NN+2, 1, 086, 079, ND, Word N12
                DATA    NX+0, 1, 069, 000
                DATA    NN+0, 1, 072, 060, ND, Word N12
                DATA    NX+0, 1, 072, 000
                DATA    NN+0, 1, 069, 060, ND, Word N12
                DATA    NX+0, 1, 069, 000, NX+2, 1, 086, 000
                DATA    NN+0, 1, 077, 072, NN+2, 1, 088, 080, ND, Word N12
                DATA    NX+0, 1, 077, 000
                DATA    NN+0, 1, 072, 060, ND, Word N12
                DATA    NX+0, 1, 072, 000
                DATA    NN+0, 1, 069, 060, ND, Word N12
                DATA    NX+0, 1, 069, 000, NX+2, 1, 088, 000
                DATA    NN+0, 1, 077, 072, NN+2, 1, 091, 082, ND, Word N12
                DATA    NX+0, 1, 077, 000
                DATA    NN+0, 1, 072, 060, ND, Word N12
                DATA    NX+0, 1, 072, 000
                DATA    NN+0, 1, 069, 060, ND, Word N12
                DATA    NX+0, 1, 069, 000, NX+2, 1, 091, 000
                DATA    NN+0, 1, 077, 072, NN+2, 1, 088, 080, ND, Word N12
                DATA    NX+0, 1, 077, 000
                DATA    NN+0, 1, 072, 060, ND, Word N12
                DATA    NX+0, 1, 072, 000
                DATA    NN+0, 1, 069, 060, ND, Word N12
                DATA    NX+0, 1, 069, 000

                DATA    NN+1, 1, 083, 074, ND, Word N04
                DATA    NX+1, 1, 083, 000
                DATA    NN+1, 1, 083, 065, ND, Word N04
                DATA    NX+1, 1, 083, 000
                DATA    NN+1, 1, 083, 065, ND, Word N04
                DATA    NX+1, 1, 083, 000, NX+2, 1, 088, 000
                DATA    NN+1, 1, 083, 045, ND, Word N04
                DATA    NX+1, 1, 083, 000

                DATA    $FF


' -----[ Initialization ]----------------------------------------------------


' -----[ Program Code ]-------------------------------------------------------

Main:
  eePntr = Mystery
  GOSUB Play_EE
  END


' -----[ Subroutines ]--------------------------------------------------------
```

```
Play_EE:
  READ eePntr, cmd                              ' get command
  IF (cmd < $FF) THEN
    SELECT cmd
      CASE $C0 TO $CF                           ' change patch
        READ (eePntr + 1), patch                ' read new patch #
        SEROUT MidiOut, MidiBaud, [cmd, patch]
        eePntr = eePntr + 2

      CASE $80 TO $8F, $90 TO $9F               ' note off or on
        READ (eePntr + 1), notes                ' read number of notes
        FOR idx = 0 TO (notes * 2 - 1)          ' read notes/velocities
          READ (eePntr + 2 + idx), midi(idx)
        NEXT
        SEROUT MidiOut, MidiBaud, [cmd, STR midi\(notes * 2)]
        eePntr = eePntr + 2 + (notes * 2)

      CASE ND                                   ' note duration
        READ (eePntr + 1), Word nTime           ' read note timing
        PAUSE nTime
        eePntr = eePntr + 3

    ENDSELECT
    GOTO Play_EE
  ENDIF
  RETURN
```