[mini-project] Create standard MIDI files with LabVIEW[*]

Ed Doering

Abstract

In this project you will create your own LabVIEW application that can produce a standard MIDI file. You will first develop a library of utility subVIs that produce the various components of the file (header chunk, track chunks, MIDI messages, meta-events, and delta times), as well as a subVI to write the finished binary file. You will then combine these into a a top-level VI (application) that creates a complete MIDI file based on an algorithm of your choosing.

1 Required Background

If you have not done so already, please study the pre-requisite modules, MIDI Messages[1] and Standard MIDI Files[2]. You will need to refer to both of these modules in order to complete this activity. Also, you will find it helpful to have already worked through the mini-project MIDI File Parsing[3].

2 Introduction

In this project you will create your own LabVIEW application that can produce a standard MIDI file. You will first develop a library of six subVIs that can be combined into a top-level VI that operates just like **MIDI_UpDown.vi** below (click the “Run” button (right-pointing arrow) to create the MIDI file, then double-click on the MIDI file to hear it played by your soundcard):

This is an unsupported media type. To view, please see
http://cnx.org/content/m15054/latest/MIDI_UpDown.llb

MIDI_UpDown.vi produces a two-track MIDI file, with one track an ascending chromatic scale and the other a descending chromatic scale. You can select the voice for each track by choosing a tone number in the range 1 to 128. You can also select the duration of each note (“on time”) and space between the notes (“off time”).

[*] Version 1.2: Mar 17, 2008 9:25 pm -0500
[†] http://creativecommons.org/licenses/by/2.0/
[1] "MIDI Messages" <http://cnx.org/content/m15049/latest/>
[2] "Standard MIDI Files" <http://cnx.org/content/m15051/latest/>
[3] "[mini-project] Parse and analyze a standard MIDI file" <http://cnx.org/content/m15052/latest/>

Remember, **MIDI_UpDown.vi** is simply a demonstration of a top-level VI constructed from the subVIs that you will make. Once you have constructed your library of subVIs, you will be able to use them in a wide array of projects; here are some ideas:

- simulated wind chime: with an appropriate voice selection (see the General MIDI Level 1 Sound Set[4] to choose a bell-like sound) and random number generator for delta times
- bouncing ping-pong ball: write a mathematical formula to model the time between bounces and the intensity of bounces
- custom ring-tone generator for a cell phone

These are just a few ideas – be creative! Remember to take advantage of your ability to control the sound type, note-on velocity, pitch bend, etc.

3 Tour of the Top-Level Block Diagram

Click the image below to take a quick tour of the top-level block diagram of **MIDI_UpDown.vi**. The role of each subVI will be discussed in some detail, and you will have a better idea of the design requirements for each of the subVIs you will create.

Image not finished

Figure 1: [video] Tour of the top-level block diagram of MIDI_UpDown.vi

4 SubVI Library

You will create six subVIs in this part of the project. **Develop them in the exact order presented!** Also, make sure you **test and debug each subVI** before moving on to the next. Many of the concepts and techniques you learn at the beginning carry forward to the more sophisticated subVIs you develop toward the end.

The requirements for the subVIs are detailed in the following sections. **Input Requirements** specify the name of the front panel control, its data type, and default value, if needed). **Output Requirements** are similar, but refer to the front panel indicators. **Behavior Requirements** describe in broad terms the nature of the block diagram you need to design and build.

An interactive front panel is provided for each of the subVIs as an aid to your development and testing. By running the subVI with test values, you can better understand the behavioral requirements. Also, you can compare your finished result with the "gold standard," so to speak.

Screencast videos offer coding tips relevant to each subVI. The videos assume that you are developing the modules in the order presented.

All right, time to get to work!

> NOTE: The file name convention adopted for this project will help you to better organize your work. Use the prefix "midi_" for the subVIs, and "MIDI_" for top-level applications that use the subVIs. This way all related subVIs will be grouped together when you display the files in the folder.

[4] http://www.midi.org/about-midi/gm/gm1sound.shtml

5 midi_PutBytes.vi

midi_PutBytes.vi accepts a string and writes it to a file. If the file already exists, the user should be prompted before overwriting the file.

5.1 Input Requirements

- file path (file path type)
- string (string type)

5.2 Output Requirements

- error out (error cluster)

5.3 Behavior Requirements

- Create a new file or replace an existing file
- If replacing a file, prompt the user beforehand to confirm
- Write the string to the file, then close the file
- Connect the file-related subVIs to `error out`

Your finished subVI should behave like this one:

This is an unsupported media type. To view, please see http://cnx.org/content/m15054/latest/midi_PutBytes.llb

5.4 Coding Tips

Watch the screencast video to learn how to use the built-in subVIs **Open/Create/Replace File**, **Write to Binary File**, and **Close File**. Refer to the module Creating a subVI in LabVIEW[5] to learn how to create a **subVI**.

Figure 2: [video] Learn how to write a string to a binary file

6 midi_AttachHeader.vi

Once all of the track strings have been created, **midi_AttachHeader.vi** will attach a header chunk to the beginning of the string to make a complete string prior to writing to a file. The header chunk requires the MIDI file type, number of tracks, and division (ticks per quarter note).

[5]"Create a SubVI in LabVIEW" <http://cnx.org/content/m14767/latest/>

6.1 Input Requirements

- string in (string type)
- type (16-bit unsigned integer type; defaults to 1)
- number of tracks (16-bit unsigned integer type; defaults to 1)
- ticks per qnote (16-bit unsigned integer type; defaults to 120)

6.2 Output Requirements

- string out (string type)

6.3 Behavior Requirements

- Create a header chunk ID sub-string (MThd)
- Create a sub-string for chunk length (always 0x00_00_00_06)
- Create sub-strings for the three unsigned integers applied as inputs
- Assemble the sub-strings into a string in order as chunk ID, chunk length, type, number of tracks, and division
- Append the inbound string to the header and output this result

Your finished subVI should behave like this one:

This is an unsupported media type. To view, please see
http://cnx.org/content/m15054/latest/midi_AttachHeader.llb

6.4 Coding Tips

Watch the screencast video to learn how to use the **Concatenate Strings** node to join substrings together into a single string. You will also learn how use the nodes **To Variant** and **Variant to Flattened String** to convert a numerical value into its representation as a sequence of bytes in a string.

Figure 3: [video] Learn how to concatenate strings and convert numerical values to a sequence of bytes

7 midi_FinishTrack.vi

Once all of the delta-time/event pairs have been assembled into a string, **midi_FinishTrack.vi** will attach a track chunk header to the beginning of the string and append an end-of-track meta-event at the end of the string. The resulting string will represent a complete track chunk.

7.1 Input Requirements

- string in (string type)
- delta-time / event pairs (string type)

7.2 Output Requirements

- string out (string type)

7.3 Behavior Requirements

- Create a track chunk ID sub-string (MTrk)
- Create a sub-string for a zero delta-time followed by an end-of-track meta-event
- Determine the total number of bytes in the track, and create a four-byte substring that represents this value
- Assemble the sub-strings into a string in order as chunk ID, chunk length, inbound string, and zero delta-time, and end-of-track meta-event, and output this result

Your finished subVI should behave like this one:

This is an unsupported media type. To view, please see
http://cnx.org/content/m15054/latest/midi_FinishTrack.llb

7.4 Coding Tips

Watch the screencast video to learn how to use the nodes **String Length** and **To Unsigned Long Integer** to determine the number of bytes in the track.

Figure 4: [video] Learn how to determine the length of string

8 midi_ToVLF.vi

midi_ToVLF.vi accepts a 32-bit unsigned integer and produces an output string that is anywhere from one to four bytes in length (recall that VLF = variable length format). You may find this subVI to be one of the more challenging to implement! Review the module Standard MIDI Files[6] to learn about variable-length format.

8.1 Input Requirements

- x (32-bit unsigned integer)
- string in (string type)

[6]"Standard MIDI Files" <http://cnx.org/content/m15051/latest/>

8.2 Output Requirements

- string out (string type)

8.3 Behavior Requirements

- Accept a numerical value to be converted into a sub-string one to four bytes in length in variable-length format
- Append the sub-string to the inbound string, and output the result

Your finished subVI should behave like this one:

This is an unsupported media type. To view, please see http://cnx.org/content/m15054/latest/midi_ToVLF.llb

8.4 Coding Tips

Watch the screencast video to learn how to convert a numerical value to and from the **Boolean Array** data type, an easy way to work with values at the bit level.

Image not finished

Figure 5: [video] Learn how to convert a numerical value to a Boolean array in order to work at the individual bit level

9 midi_MakeDtEvent.vi

midi_MakeDtEvent.vi creates a delta-time / event pair. The subVI accepts a delta-time in ticks, a MIDI message selector, the channel number, and two data values for the MIDI message. The delta-time is converted into variable-length format, and the (typically) three-byte MIDI message is created. Both of these values are appended to the inbound string to produce the output string. Review the module MIDI Messages[7] to learn more.

9.1 Input Requirements

- string in (string type)
- delta time (32-bit unsigned integer; defaults to 0)
- event (enumerated data type with values Note Off, Note On, Control Change, Program Change, Pitch Wheel; defaults to Note Off)
- channel (8-bit unsigned integer; defaults to 1)
- data 1 (8-bit unsigned integer; defaults to 0)
- data 2 (8-bit unsigned integer; defaults to 0)

[7]"MIDI Messages" <http://cnx.org/content/m15049/latest/>

9.2 Output Requirements

- string out (string type)

9.3 Behavior Requirements

- Convert the delta time value to VLF format using the **midi_ToVLF.vi** subVI you created in a previous step
- Subtract 1 from the inbound channel number (this way you can refer to channel numbers by their standard numbers (in the range 1 to 16) outside the subVI.
- Create a MIDI message status byte using the channel number and event selector
- Finish the MIDI message by appending the appropriate byte values; depending on the MIDI message you need to create, you may use both 'data 1' and 'data 2', or just 'data 1' (Program Change message), or you may need to modify the incoming data value slightly (for example, outside the subVI it is more convenient to refer to the tone number for a Program Change message as a value between 1 and 128)
- Append the delta-time sub-string and the MIDI message sub-string to the inbound string, and output the result

Your finished subVI should behave like this one:

This is an unsupported media type. To view, please see http://cnx.org/content/m15054/latest/midi_MakeDtEvent.llb

9.4 Coding Tips

Watch the screencast video to learn how to assemble a byte at the bit level, and also how to set up the enumerated data type for a case structure.

Figure 6: [video] Learn how to assemble a byte at the bit level, and learn how to set up an enumerated data type for a case structure

10 midi_MakeDtMetaEvent.vi

midi_MakeDtMeta.vi creates a delta-time / meta-event pair. The subVI accepts a delta-time in ticks, a meta-event selector, text string, and tempo value (only certain meta-events require the last two inputs). The delta-time is converted into variable-length format, and the meta-event is created. Both of these values are appended to the inbound string to produce the output string. Review the module Standard MIDI Files[8] to learn about meta-events.

[8]"Standard MIDI Files" <http://cnx.org/content/m15051/latest/>

10.1 Input Requirements

- string in (string type)
- delta time (32-bit unsigned integer; defaults to 0)
- event (enumerated data type with values Text, Copyright Notice Text, Track Name, Instrument Name, Lyric Text, Marker Text, Cue Point Text, Sequencer-Specific, End of Track, and Set Tempo; defaults to Track Name)
- text (string type)
- tempo (32-bit unsigned integer; defaults to 500,000)

10.2 Output Requirements

- string out (string type)

10.3 Behavior Requirements

- Convert the delta time value to VLF format using the **midi_ToVLF.vi** subVI you created in a previous step
- Create a meta-event sub-string using the sequence 0xFF (indicates meta-event), meta-event type (refer to a table of meta-event type numbers), meta-event length (use **midi_ToVLF.vi** for this purpose), and meta-event data.
- Append the delta-time sub-string and the MIDI message sub-string to the inbound string, and output the result

Your finished subVI should behave like this one:

This is an unsupported media type. To view, please see http://cnx.org/content/m15054/latest/midi_MakeDtMetaEvent.llb

10.4 Coding Tips

At this point you should have enough experience to proceed without assistance!

11 Top-Level VI Application

Congratulations! You now have assembled and tested six subVIs that can form the basis of many other projects. Now use your subVIs to build an **application VI** (top-level VI) to demonstrate that your subVIs work properly together. Two applications that you can easily build are described next.

11.1 Sweep Through Notes and Tones

The application block diagram pictured below produces a single-track MIDI file containing an ascending chromatic sweep over the entire range of note numbers (0 to 127). Before sounding the note, the Program Number (tone or voice selection) is set to the same value as the note number. Thus, the voice changes for each note, adding additional interest to the sound. The note duration is specified by a control whose unit is milliseconds. As an exercise, you will need to complete the grayed-out area to convert the units of "duration" from milliseconds to ticks.

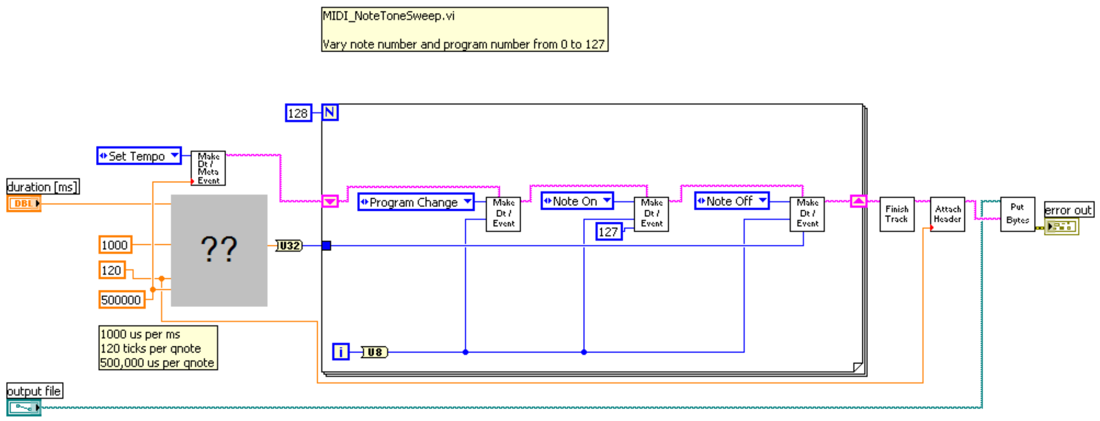

Figure 7: Block diagram to produce a single-track MIDI file containing an ascending chromatic sweep over the entire range of note numbers (0 to 127)

11.2 Measure the Velocity Profile of Your Soundcard

The application block diagram pictured below creates a MIDI file in which the same note is played repeatedly, but the velocity varies from the maximum to the minimum value in unit steps. When you play the MIDI file, you can record the soundcard's audio output and measure its **velocity profile**, i.e., the mapping between the note's velocity value and its waveform amplitude. The Audacity[9] sound editing application works well for this purpose; choose "Wave Out Mix" as the input device to record the soundcard's output.

[9]http://audacity.sourceforge.net/

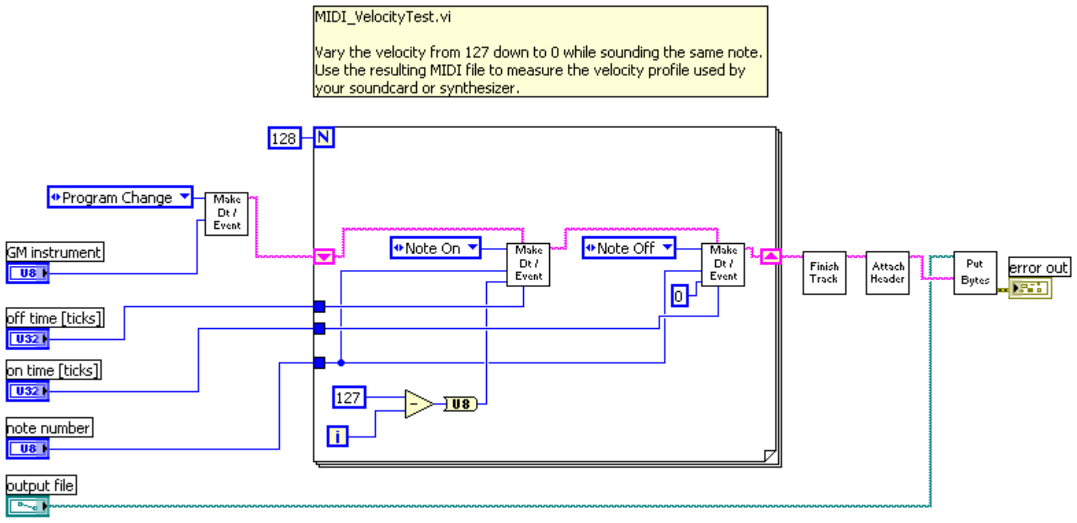

Figure 8: Block diagram to play a single note with velocity varied from 127 down to 0