Introduction

Yamaha’s WX5 Wind Midi Controller (Figure 1) is a device designed to give native wind players the ability to control MIDI instruments in a setting that is more natural and more expressive than that of other MIDI controllers. In particular, it allows for embouchure control and breath control, two abilities that are not normally found on other MIDI controllers.

After a basic search of the Internet and various Pure Data (PD) message lists, there seems to be an absence of a good template for handling Yamaha’s WX5 Wind MIDI Controller in PD. This article will focus on the development of a basic template for using the WX5 with PD.

There are three main qualities of the WX5 that need to be implemented: Program Change, Pitch Bend, and Breath Control. After a brief introduction to PD, we will go through a basic overview of the PD patch, and then move on to exploring these particular subjects in greater detail.

I. Introduction to Pure Data

A PD document is known as a “patch”. Patches are made up of various types of boxes, connected by wires (Figure 2). Boxes have inputs and/or outputs, as represented by bold lines on the top and/or bottom of the box, respectively. Wires connect inputs and outputs and transmit information from box to box. It is important to note that in this respect, PD is an implementation environment rather than a programming language. All of the objects represent smaller programs written in the programming language “C”. PD allows simple, painless implementation of these objects, as well as customization for those familiar with C (Puckette).

Figure 2 is the classic “Hello World” programming example. This particular patch includes three different types of boxes: the number box, the message box, and the object [print]. Each box plays its own particular role in the process. The [print] object is the most important part of this patch. [print] takes any message from its input and sends it out to PD’s terminal window, to be displayed for the user. The message box containing the text “Hello World” simply sends out its message (the text) through its output. Therefore, when the patch is running and
the user clicks on the message box, it sends the message “Hello World” to [print], which in turn sends the message on to the terminal window.

The number box plays a similar role to that of the message box: it simply outputs the number that is inside it. The user must click and hold on the number box, and drag the mouse up or down the change the number inside. Every time the number changes, the number is sent to [print] which sends it on to the terminal window. This small patch is a representation of all the functions and happenings that take place inside a PD patch.

An explanation of “subpatches” and “abstractions” must be given in order to ensure complete comprehension of the WX5 control patch. “Subpatches” are small patches within the main patch whose content resides within the main patch. On the other hand, “abstractions” are objects that call separate “.pd” documents. This allows for increased portability of particularly useful routines like the abstractions [windctrl] and [bendctrl]. While they appear to be normal PD objects, abstractions are actually calling separate PD documents.

II. Basic Overview

The WX5 control patch begins with the object [notein] (Figure 3). This object receives a MIDI message from the controller (in this case, the WX5) and outputs a note number and velocity pair. From there, this note/velocity pair is routed out to a synthesizer. The synthesizer chosen will depend on the program number transmitted by the controller. A program number must be selected on the controller for the patch to know where to send the notes. After being converted to frequency and amplitude in the synthesizer, the note is then routed back to the main output. From here, it is scaled by the abstraction [windctrl], scaled to avoid clipping, and then sent out to the user’s audio hardware.

III. Program Change

The section of the patch that handles program changes is actually quite simple, but incredibly important. As shown in Figure 4 (inside subpatch [pd synthpatcher1]), the patch receives program numbers from the controller and attaches the numbers to note/velocity pairs via [pack]. The notes are then sent off to the synthesizers (subpatches called pd program1, pd program2, etc.) where they are eventually routed by program number. The program number dictates to which synthesizer program the note/velocity pair will be sent.

IV. Synthesizers

Once the note/velocity pair has been sent to a program, it is then processed by the synthesizer (Figure 5). The synthesizer handles both pitch and volume. The note/velocity pair
is split up by the object [unpack] and routed in separate directions. The note is sent to the abstraction [bendctrl] where it is converted from MIDI note (a number from 0-127) to frequency, and the effects of the bend-wheel on the MIDI controller are applied. This is also where the effects of embouchure adjustments will be applied, as embouchure changes are transmitted on the bend controller. From there, the frequency travels out to an oscillator, object [osc~], that creates a sine wave at the specified frequency. This oscillator can be replaced with any other kind of wave generator, or a combination. The wave generator chosen will determine the timbre of the sound.

On the other side, velocity is transformed into volume. Velocity numbers, like MIDI notes, are on a scale from 0-127, 0 being off, and 127 being the loudest. The velocity side of the subpatch takes that range of numbers and maps it out to range between 0 and 1. This number is then scaled by [line~] to remove clicks, and then is combined with the frequency and sent on to the main output.

This process is done simultaneously in all 16 synthesizer subpatches to produce polyphony.

V. Output and Breath Control

As seen in Figure 3, the object [catch~] collects the signals from all of the synthesizers and sends them on to the main output. However, before they get there, they are first scaled by the abstraction [windctrl]. See Figure 6. This abstraction takes advantage of the WX5’s ability to control volume with the breath of the player. The principle of [windctrl] is nearly identical to that of the earlier volume control. Controller data (numbers 0-127) is received from the breath controller, controller 7. The numbers are then mapped to a range of 0 to 2 (rather than 0 to 1, for increased dynamic range) and then sent out to be multiplied with the overall volume. This allows the player to control the overall volume with his or her breath. From this point, the sounds travel through the digital to analog converter (the [dac~] object) and out to the user’s hardware.

VI. Uses and Expandability

This patch is a great template for someone interested in using their Wind MIDI Controller in conjunction with PD. Although it was originally designed for Yamaha’s WX5, it can be easily modified to accept other similar controllers with some slight alterations. The template allows for easy implementation and/or addition of other synthesizer sounds and patches to suit the user’s needs. Directions for creating new sounds and adding programs are documented within the patch.

Anyone interested in acquiring the patch can email me at acjones@lagrange.edu.
Wind MIDI Control in Pure Data

Works Cited
WX5 Wind Midi Controller Owner’s manual PD Documentation: http://www-crca.ucsd.edu/~msp/Pd_documentation/

Acknowledgments
I’d like to thank Dr. Mitchell Turner for his help with this project, especially in troubleshooting. Also, I offer my highest gratitude to the Undergraduate Research Committee for funding this project.

![Figure 1: WX5 Wind MIDI Controller](image1)

![Figure 2: Hello World](image2)
Figure 3: Main Patcher Window

- **noin**: Receive note-velocity pairs from controller
  - Note out left outlet, velocity out middle outlet
- **pack 0 0**: Combine note and velocity into a list
- **poly 16**: Manages the polyphony (max 16 notes)
  - Sends out voice-number, MIDI-note, velocity
- **pack 0 0 0 0**: Combine voice-number, prog change, MIDI-note, and velocity into a list
- **pd synthSubPatchOut1**: This sends to synthesizers subpatch
- **catch= main0Output**: Receive signals from synthesizers subpatch
- **windctrl**: Uses ctrlin 7 from WX5 to scale audio output
  - ***=**: Scale summed signal to eliminate "samples out of range"
  - **clip~ -1 1**: Make sure final output does not distort
  - **dac~**: Send the signal out to the audio hardware

Figure 4: Program Change

- **inlet**: Bring in Program, MIDI-note and Velocity
- **route 1 2 3 4**: Route by program number
- **pd program4**, **pd program3**, **pd program2**, **pd program1**: Each program is a different synthesizer
Figure 5: Synthesizer Innards

Inlet <- bring in MIDI note and velocity

Unpack 0 0

Pitch Side

Outputs freq => bendctrl

Sine wave => osc~

This is where the quality of the wave can be changed

Velocity Side

Velocity from 0 to 127 is squared and then divided by 16129 to range between 0 and 1

/ 16129

Pack f 10

Line~ <= remove clicks, changes to the current velocity in 10 milliseconds

<= scale the output

Throw~ main0output <- send audio signal to [catch~ main0output] found in the main patcher window

Figure 6: [windctrl] abstraction

[This handles breath control, controller 7]

cctlin 7

t b f square, divide by 127 squared to provide a number between 0 and 1

* 16129

* 2 <= 2 multiplier to make change more noticeable

Pack f 10

Line~ <= remove noise

Outlet~