Computer Music Systems and Information Processing

Roger B. Dannenberg, instructor
Spring 2019

Course Overview

- Systems
  - Sensors → Control → Synthesis
  - MIDI representation, I/O, timing, files, …
  - Scheduling, synchronization, networks
- Information Processing
  - Music representation and manipulation
  - Automatic score following and accompaniment
  - Pattern matching and music database search
  - Automatic music generation

Expectations

- Syllabus is on-line
  - www.cs.cmu.edu/~music/cmsip
  - There are lists of concepts for every week. My goal is to teach these – if you don't feel you've mastered basic knowledge pertaining to these concepts, one of us is not doing their job.
- Readings – everything on-line. No textbook.
- Deliverables
  - Six (6) programming assignments.
  - Five (5) short-answer homework on readings.
  - Exams – midterm and final
- Lectures – attendance is required.
- Concerts – TBD, probably STUDIO for Creative Inquiry (CFA)

TAs

- Shuqi Dai
- Caroline Wu

- see http://www.cs.cmu.edu/~music/cmsip or http://piazza.com/ for contact info, office hours, etc.

Grading

- Programming assignments: 40%
- Participation: 10%
- Other homework: 15%
- Midterm: 15%
- Final Exam: 20%

Some Things We Will Do

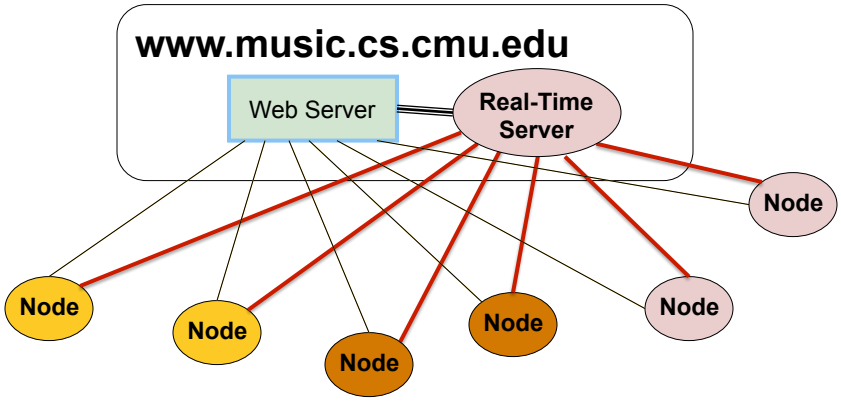
- Write programs to:
  - Plays sounds via MIDI
  - Implement a music sequence player
  - Implement live audio processing in C
  - Generate music automatically
  - *Play music! End of semester concert.*
- Learn about
  - Music theory
  - Music representation
  - Music information retrieval

Demo

Some hacks with Serpent,
MIDI, Soundcool, and O2

Internet Drum Circle (maybe)

An extra project for 15-623, optional for others.



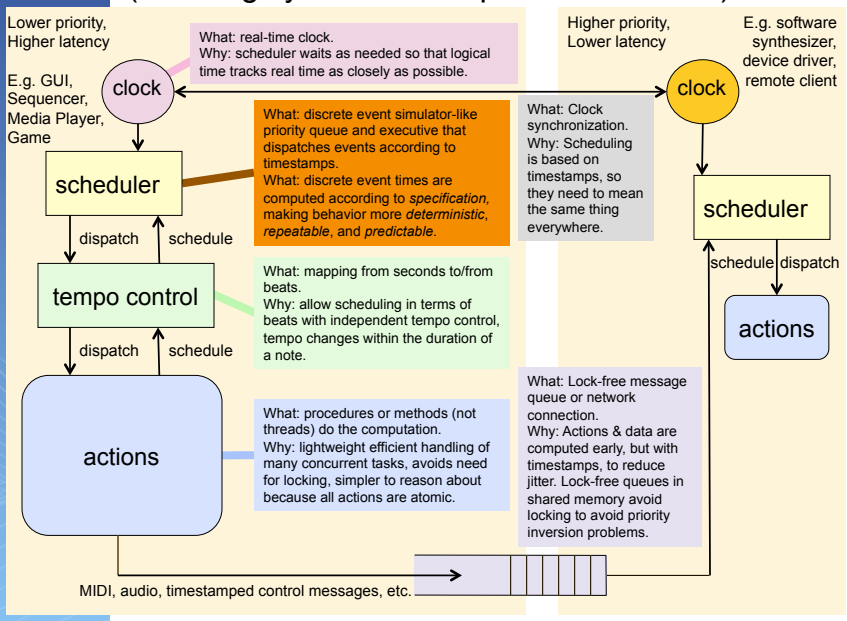
Concert: Interactive Group Music Performance

- All-class Laptop Orchestra
- Develop class projects into something you can perform
- Interactive control, but
- Synchronized and conducted over network
- Another option might be developing a piece with the Exploded Ensemble (a course offered by the School of Music)

Concert Example: Spring 2017 CMSIP Laptop Ensemble

(Video)

The Big Picture
(covering systems concepts in this course)



Survey: Please complete this survey: https://goo.gl/forms/N9fADlBS0BHxoaEE3

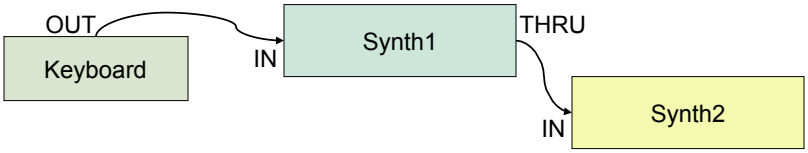
- Name & Andrew ID
- Year: Fresh, Soph, Junior, Senior, Grad (year)
- Musical Background:
  - Play an instrument? (barely, yes, good, pro)
  - Read music? (barely, yes, good, pro)
  - Use MIDI (never, some, lots)
- Programming Background:
  - Name of most advanced software implementation course (including this semester)
  - Write languages you have used: Java, C, C++, Python, ML, Matlab
- Main interest (pick one):
  - Automatic Music Generation (Algorithmic Composition)
  - Music Information Retrieval (search, classification, ...)
  - Music Understanding (beat detection, chord recognition, ...)
  - Other: _____
- I have a laptop with an x64 processor and sound: yes / no

Let's Get Started

- Quick Intro to MIDI
- Software for homework/projects
- PortMidi – your friendly MIDI API
- Serpent

MIDI

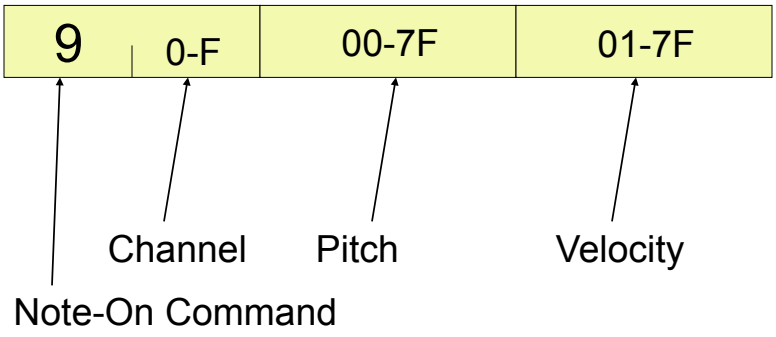
- Carries controls from keyboards to synthesizers.



- Commands correspond to physical controller changes:
  - Key Down
  - Key Up
  - Program Change
  - Pitch Bend
  - Volume Pedal

Note-On in Detail

- 3 bytes:



PortMidi

- Cross-platform API
- C Programming Language
- For MIDI Input/Output

- Includes PortTime
  - Cross-platform API for real-time clock
  - Facility to call a function periodically
- PortTime inspired by PortAudio
- PortMusic: PortAudio, PortMidi, PortSMF, …

Getting Started

- You can use PortMidi with C++
  - Download PortMidi
  - Try some applications in pm_test
- But in this class, use Serpent (described later)
  - PortMidi is built-in
  - Try: load "midi_out_test"
- Full PortMidi documentation is:
  - pm_common/portmidi.h
  - on the web at Sourceforge

Serpent Interface to PortMidi

- **What's Serpent? Let's worry about it later.**

```
midi_create() => port          midi_close(port)
midi_in_default() => devno     midi_abort(port)
midi_out_default() => devno    midi_poll(port) => n
midi_count_devices() => n      time_start(resolution)
midi_get_device_info(devno)    time_get()=>float_seconds
  => ["interface_name",
      "device_name", input?, output?]
midi_open_input(port, devno, buffer_size)
midi_open_output(port, devno, buffer_size, latency)
midi_read(port)=> [time, msg]
midi_write(port, time, msg)
```

Library Initialization

- `PmError Pm_Initialize( void );`
  Initializes PortMidi library. Usually do this once.
- `PmError Pm_Terminate( void );`
  Close library and release resources.
- `const char *Pm_GetErrorText( PmError errnum );`
  Translate an error code to a printable string.
- `void Pm_GetHostErrorText(char * msg, unsigned int len);`
  If a function returns an error code of pmHostError, the error is host-specific. Get an ascii representation (to print for the user) by calling this function.

Finding a Midi Device

- `int Pm_CountDevices( void );`
  How many devices (MIDI ports) are there?
  Devices are numbered from 0 to N-1.
- `const PmDeviceInfo* Pm_GetDeviceInfo( PmDeviceID id );`
  Get info for a particular device:

```
typedef struct {
  int structVersion;
  const char *interf; /* underlying MIDI API, e.g. MMSystem */
  const char *name;   /* device name, e.g. USB MidiSport 1x1 */
  int input; /* true iff input is available */
  int output; /* true iff output is available */
  int opened; /* used by generic PortMidi code to do error checking */
} PmDeviceInfo;
```

- `PmDeviceID Pm_GetDefaultInputDeviceID( void );`
- `PmDeviceID Pm_GetDefaultOutputDeviceID( void );`

Opening MIDI for Output

```
PmError Pm_OpenInput( PortMidiStream** stream,
                      PmDeviceID inputDevice,
                      void *inputDriverInfo,
                      long bufferSize,
                      PmTimeProcPtr time_proc,
                      void *time_info );

PmError Pm_OpenOutput( PortMidiStream** stream,
                       PmDeviceID outputDevice,
                       void *outputDriverInfo,
                       long bufferSize,
                       PmTimeProcPtr time_proc,
                       void *time_info,
                       long latency );
```


Sending a MIDI Message

- ```
  #define Pm_Message(status, data1, data2) \
          ((((data2) << 16) & 0xFF0000) | \
           (((data1) << 8) & 0xFF00) | \
           ((status) & 0xFF))
  ```
  Messages are encoded in longs (32-bit integers), e.g. Pm_Message(0x90, 60, 100)

- ```
  PmError Pm_WriteShort( PortMidiStream *stream,
        PmTimestamp when, long msg);
  ```
  Use a timestamp of zero to send immediately (or after latency – see Pm_OpenOutput)

Receiving a MIDI Message

```
typedef struct {
   PmMessage       message;
   PmTimestamp     timestamp;
} PmEvent;
```

Data is timestamped.

```
PmError Pm_Read( PortMidiStream *stream,
          PmEvent *buffer, long length );
```

Returns number of events read or error code (errors are negative, counts are non-negative).

```
PmError Pm_Poll(PortMidiStream *stream);
```

Returns number of events ready to be read or error code.

Week 1, Day 2

SERPENT AND MIDI

Serpent

- Scripting language based on Python
- Cross-platform
- Open-source C++
- Real-time garbage collection
- Built-in timing, MIDI, threads, GUI, and networking you can use for projects in this course

Serpent Documentation

- Mostly on one page – see serpent/doc.
- Separate (small) pages for:
  - PortMIDI extensions
  - Proc extension (implements periodic callbacks and message queues)
  - Network extension (implements simple client/ server communication through TCP/IP)
  - ZeroMQ (another networking layer)
  - O2, an extension of OSC

Serpent Syntax and Semantics

- From Python:
  - Indentation (not brackets) for program structure
  - Newlines for statement separators (semicolons are optional)

```
def absolute_value(x):
    if x >= 0
        return x
    else
        return -x
```


Alternative Styles

```
def absolute_value(x)
    if x >= 0
        return x
    else
        return -x
```

```
def absolute_value(x):
    if x >= 0:
        x
    else:
       -x
```

```
def absolute_value(x):
    if x >= 0:
        return x;
    else:
        return -x;
```

```
def absolute_value(x):
    if x >= 0: return x
    else:      return -x
```

```
def absolute_value(x):
    x if x >= 0 else -x
```

Data types

- String (immutable): “hello world”
- Atom (unique strings): ‘hello world’
- Int (50-bit signed integers): 57, 0x00fe
- Double (64-bit floating point): 57.0
- Arrays (1-dim, dynamic):
  [‘an’, ‘array’, 1, 2, 5.2]
- Dictionaries: { ‘foo’ : 1, ‘bar’ : 34,
  ‘baz’ : “cmu”, 17: -0.123}
- Objects (user defined)
- Boolean: t, nil, true, false, non-nil -> t, 0 -> t

Serpent Function Definition

```
def my_function(p1, p2, p3):
    # this is a comment
    // this is a comment too
    var local_variable
    var local2 = 6.5
    not_a_local = "a string“
    …
    local2 // last expression is returned
```

Basic Control Constructs

```
if expr1
    statement(s)
elif expr2
    statement(s)
else
    statement(s)

for i = 0 to 10:
    statements

for elem in array_expr:
    statements

while expr
    statements
```

```
load "filename"

require "filename"

print expr, expr; expr,

display "label", expr, expr, …
```

```
funcall(), apply(), send(),
   sendapply()
```

Example:

```
def foo(a, b)
    display "foo", a, b
apply('foo', [1, 2])
```

➔ `foo: a = 1, b = 2`

Defining and Using a Class

```
class Account:
    var balance
    def init(initial_deposit)
        balance = initial_deposit
    def deposit(x)
        balance = balance + x
    def withdraw(x)
        if balance >= x
            balance = balance - x
            return balance
        else
            return false
```

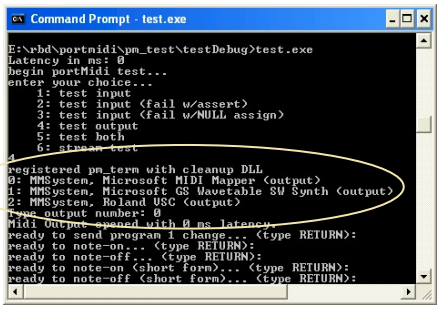
```
// make an instance
account = Account(0)

// call a method
account.deposit(5)

// access an instance variable
print account.balance

// use/test return value
if not account.withdraw(10):
    print "don't have $10"
```

Example from midi_out_test.srp

```
MIDI_ID = midi_out_default()
def midi_out_test():
    var mo = midi_create()
    // parameters are midi object, device ID, number of buffers, and
   latency in ms
    var result = midi_open_output(mo, MIDI_ID, 100, 0)
    if (result != 0):
        print "midi_open returns "; result
        return
    time_started = time_get() // prepare time_elapsed()
    print "time test start"
    wait_until(2.0)
    print "time test end"
    midi_note_on(mo, 0, 60, 100)
    wait_until(4.0)
    midi_note_on(mo, 0, 62, 100)
    wait_until(6.0)
    midi_note_on(mo, 0, 60, 0)
    midi_note_on(mo, 0, 62, 0)
    result = midi_close(mo)
    if (result != 0)
        print "midi_close returns "; result
```

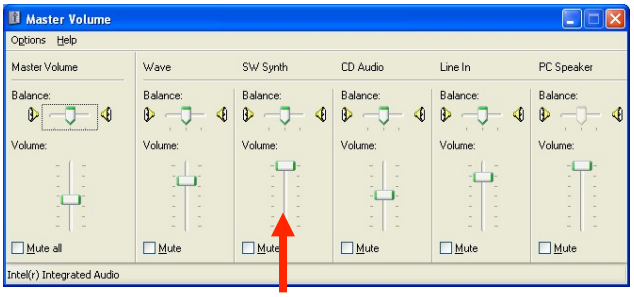

MIDI devices, pseudo-devices, and software synthesizers

- Windows has an output called MIDI Mapper that just sends MIDI to the Microsoft GS Wavetable Synth device (maybe you can change this in the registry, but there is no control panel for MIDI settings.)
- Windows comes with a built-in MIDI synthesizer called Microsoft GS Wavetable SW Synth



Don't Forget Audio Setup

- Check for any audio—play a sound file
- Check for SW Synth volume and mute – it may only appear when you are running a program that opens a software synthesizer



Setup on the Mac

- Use IAC Driver to talk to MIDI devices
  - find Audio Midi Setup (an OS X application)
  - Window : Show MIDI Window (menu item)
  - Click IAC Driver, then click "show info"
  - Add a port if you have none
- SimpleSynth from pete.yandell.com works
- SimpleSynth and Serpent must both connect to "IAC Driver Bus 1"
- You must leave SimpleSynth running!
- Rather than IAC Driver, you can send direct to SimpleSynth and tell SimpleSynth to listen to "SimpleSynth Virtual Input"

Installing Serpent

- Find zip file on class website
- Copy and expand
  - Installer for 64-bit Windows
  - Zip file with applications and libraries for 64-bit Mac OS X
  - Zip file with sources for Linux

SERPENTPATH: Mac and Linux

- TCSH:

```
setenv SERPENTPATH /home/rbd/serpent/programs:
/home/rbd/serpent/lib:/home/rbd/serpent/wxslib
```

- BASH:

```
SERPENTPATH="/Users/rbd/serpent/programs:/home/
rbd/serpent/lib:/Users/rbd/serpent/wxslib"

export SERPENTPATH
```

Homework 1

- See Syllabus
- Due Jan 22

Project 1

- Given a pre-defined *tempo* (beats per minute) and a repeat count *N*, either from command line or through a (simple please) graphical interface, the program will play *N* repetitions of a 4-beat drum pattern.

Beats(s)	1		2		3		4		1		2		…
High-hat	X	X	X	X	X	X	X	X	X	X	X	X	…
Snare Drum			X				X				X		…
Bass Drum	X				X				X				…

Approach to Project 1

- Keep it simple:
- Represent pattern as a 2D array
  - If you wish to allow multiple patterns, you can put each in a loadable file (you can even prompt for a file to load and load it)
- *Sequentially:* send note-off for previous beat and send note-on for current beat
- Wait for next beat using time_sleep(dur)
- There are many obvious shortcomings here – you will hopefully understand them, and we'll address them in future projects.

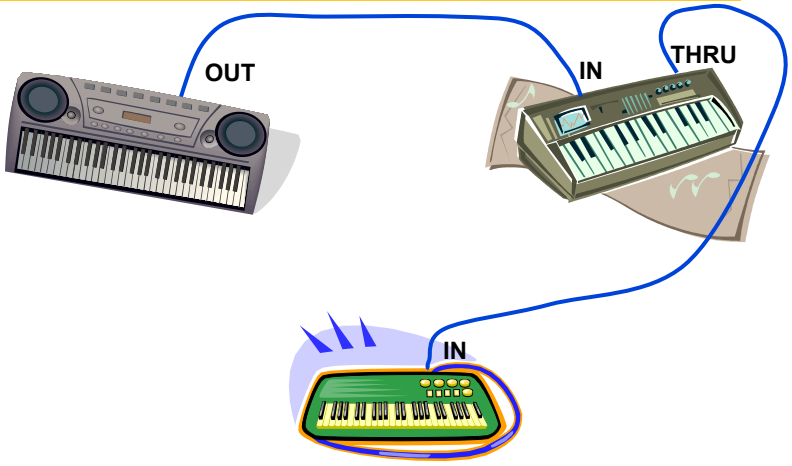
The MIDI Standard

Roger B. Dannenberg
Professor of Computer Science, Art & Music
Carnegie Mellon University

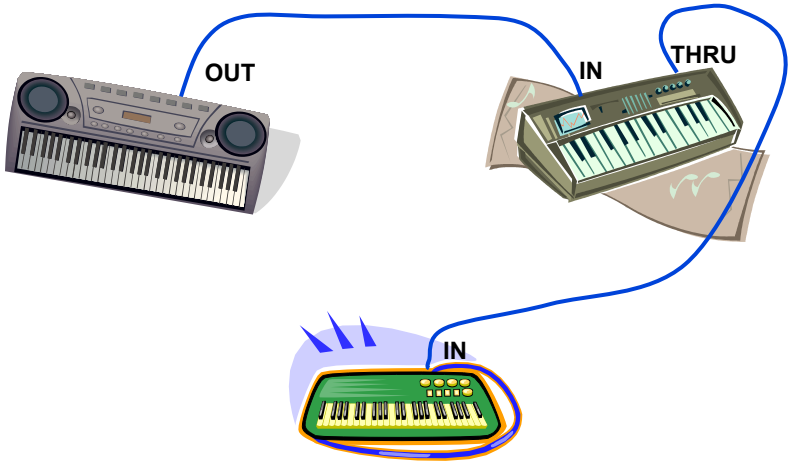
MIDI: Musical Instrument Digital Interface

- Musical Performance Information:
  - Piano Keyboard key presses and releases
  - "instrument" selection (by number)
  - sustain pedal, switches
  - continuous controls: volume pedal, pitch bend, aftertouch
  - very compact
    (human gesture < 100Hz bandwidth)

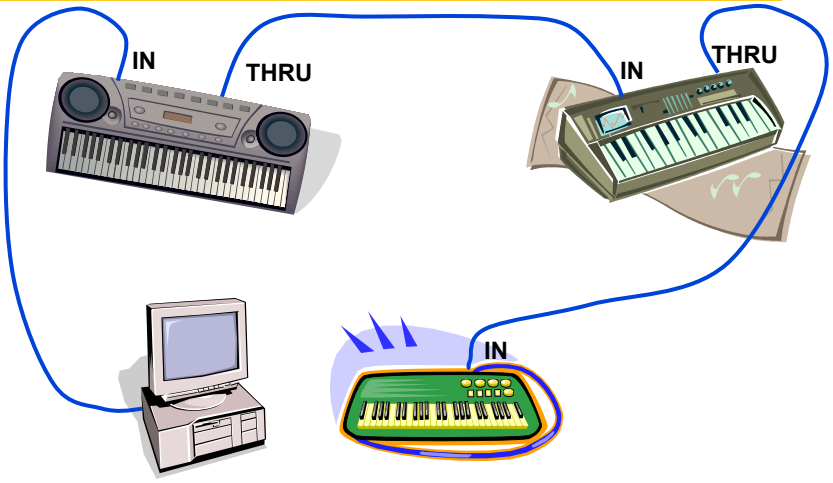
Point-to-Point Connections



Channels

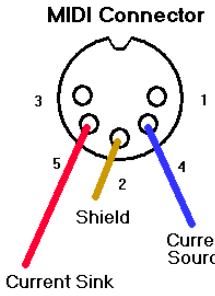


No Time Stamps



Hardware Details

- Asynchronous serial
- 8-bit bytes+start+stop bits
- 31.25K baud = 1MHz/32
- 5-pin DIN Connector
- Current loop



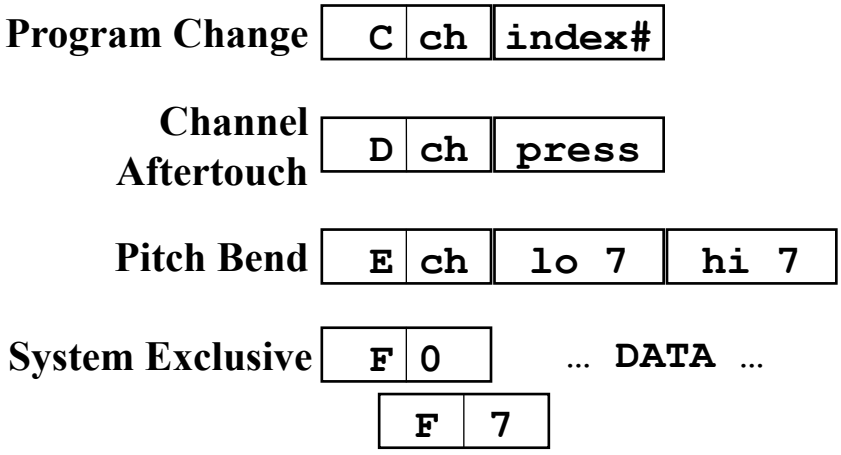
From www.colomar.com/ Shavano/ midi_cable.html

From www.answers.com/ topic/din-connector

MIDI Message Formats - 1

Key Up	8	ch	key#	vel
Key Down	9	ch	key#	vel
Polyphonic Aftertouch	A	ch	key#	press
Control Change	B	ch	ctrl#	value

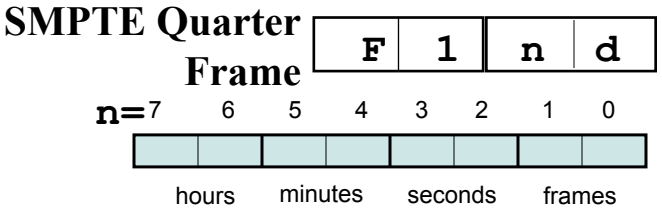
MIDI Message Formats - 2



MIDI Message Formats - 3

Timing Clock	F 8
Start	F A
Continue	F B
Stop	F C
Active Sense	F E
System Reset	F F

MIDI Message Formats - 4



More SMPTE operations are encoded into SysEx messages.

Serpent Debugger

```
load "debug"
def bug(a)
    print a + b
def main()
    bug(3)
main()
```

```
runtime exception handler called
exception is: global variable is
  unbound - b
frame variables: {'a': 3}
frame pc: 3
frame method: bug
frame class: nil

global variable is unbound - b,
  debugger invoked.
Method: bug , PC: 2 , Line: 17 ,
  File: <stdin>
1>
```

Serpent Debugging

```
load "debug"
def bug()
    print a + b
def main()
    bug()
main()
```

```
1> ?
debugger reads ?
Class nil , Method bug
Class nil , Method main
Class nil , Method
                <immediate command 0>
Class nil , Method <callin_method>
Class nil , Method <global>
1> a
debugger reads a
dbg_variable name = a
a is a local
a = 3
1> >
debugger reads >
Resume execution...
>
>
```


Serpent: Spaces vs. Tabs

- White space is significant
- Editors play games with tabs (4 cols? 8?)
- Try to turn off auto-insertion of tabs
- *Never* use tabs in Serpent code
- (Technically you can, but even *I've* forgotten the details.)

Summary and Self Assessment

- Here's what you should know now …
- Course basics:
  - Web site: exams, concerts, acad. integ. policy
  - Grades (only) will go on Blackboard
  - Q&A (only) on Piazza
- What is MIDI?
  - Message types (don't memorize constants)
- Serpent
  - How to install
  - How to read documentation and debug
  - Able to write simple functions, start Project 1

Final Reminders

- Get started immediately:
  - Install and test software
  - Get MIDI playback working (software, hardware, speakers, headphones, …)
  - See Project 1 description on website.
- Homework is due Jan 22
- Project 1 is due Jan 29