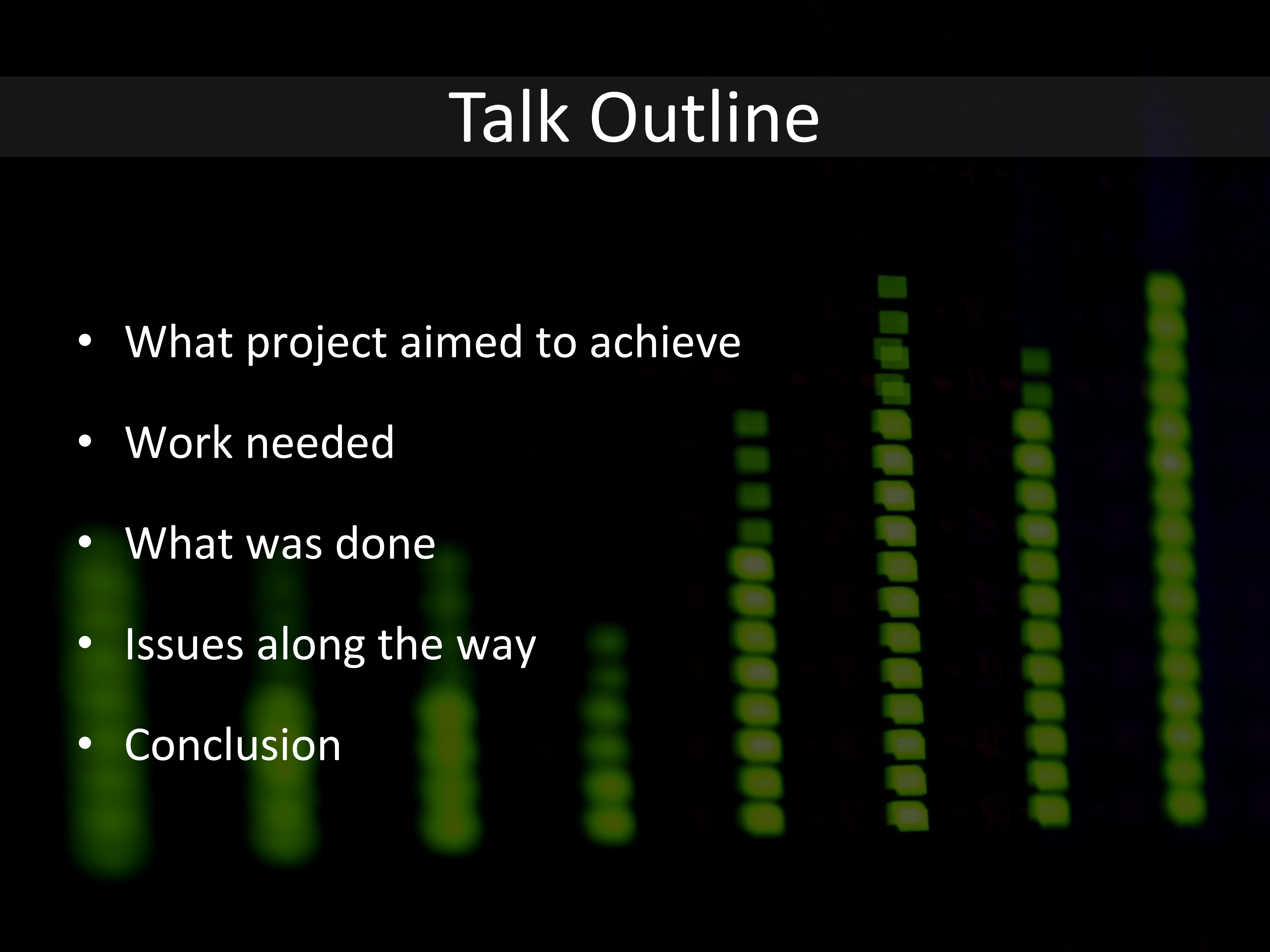
Touch screen control for digital mixing consoles

Brent Shaw



Talk Outline

- What project aimed to achieve
- Work needed
- What was done
- Issues along the way
- Conclusion

Project goal

To investigate current method of remotely controlling digital audio mixing consoles.

To develop a cross-platform version of the distributed control system MIDINet in order to allow for mobile control of audio mixing consoles.

Work to be done

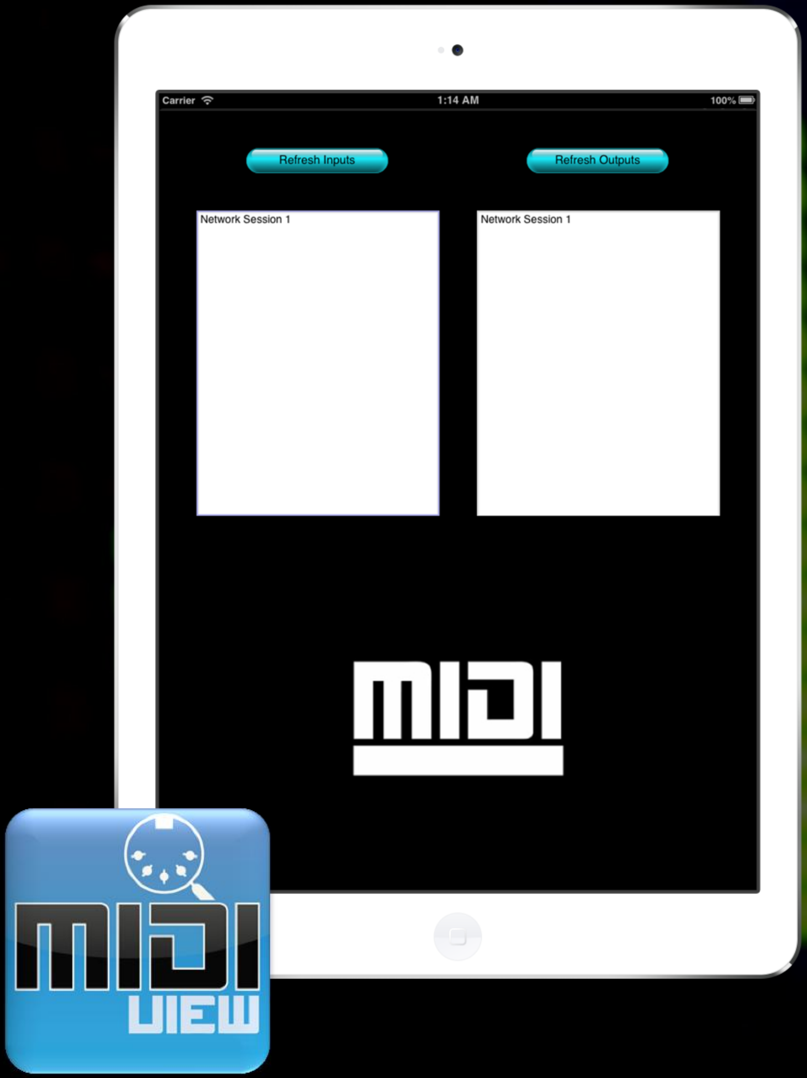
- Create a cross-platform library for MIDI device control
- Re-design MIDINet's current networking class
- Create new GUI for cross-platform application
- Update MIDINet to use JUCE rather than MFC library functions

JMidi

- MaxMidi
  - MIDI device control library for Visual C++
- JUCE provides extensive audio libraries
- Integration into MIDINet would be slow
- Library that MIDINet classes can inherit from

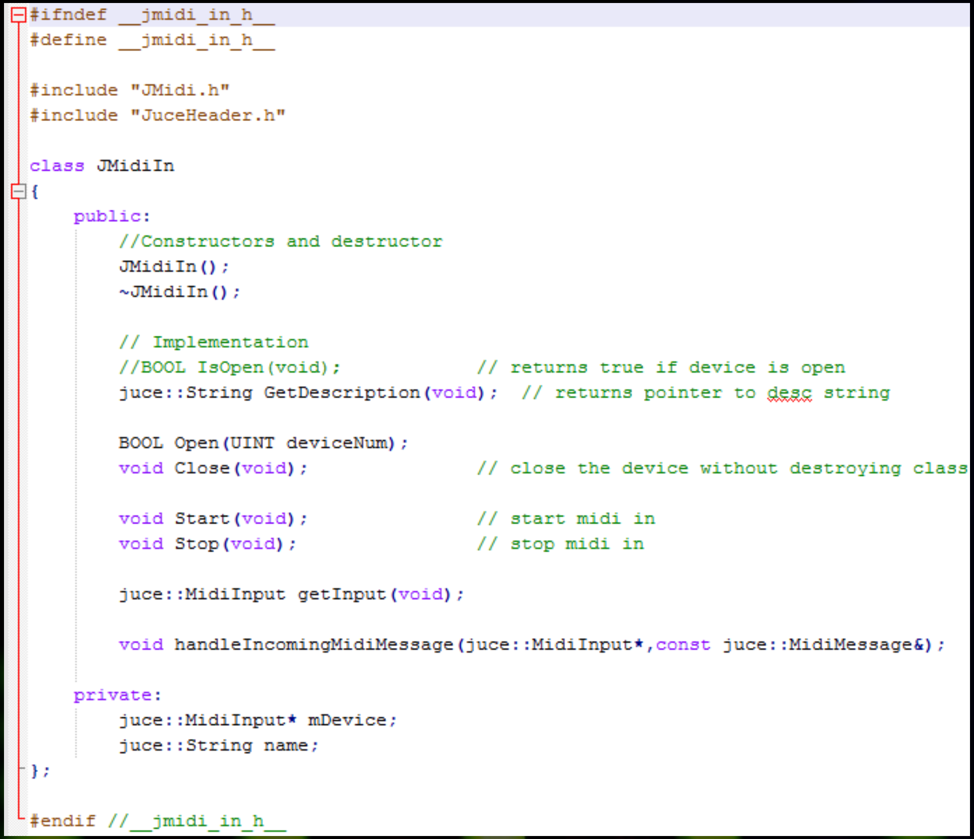
MIDIView

- An application for testing JUCE's MIDI device libraries.
- Reads attached devices as well as software/virtual ports.
- Tested on Linux, Windows, OSX and iOS.



JMidiIn

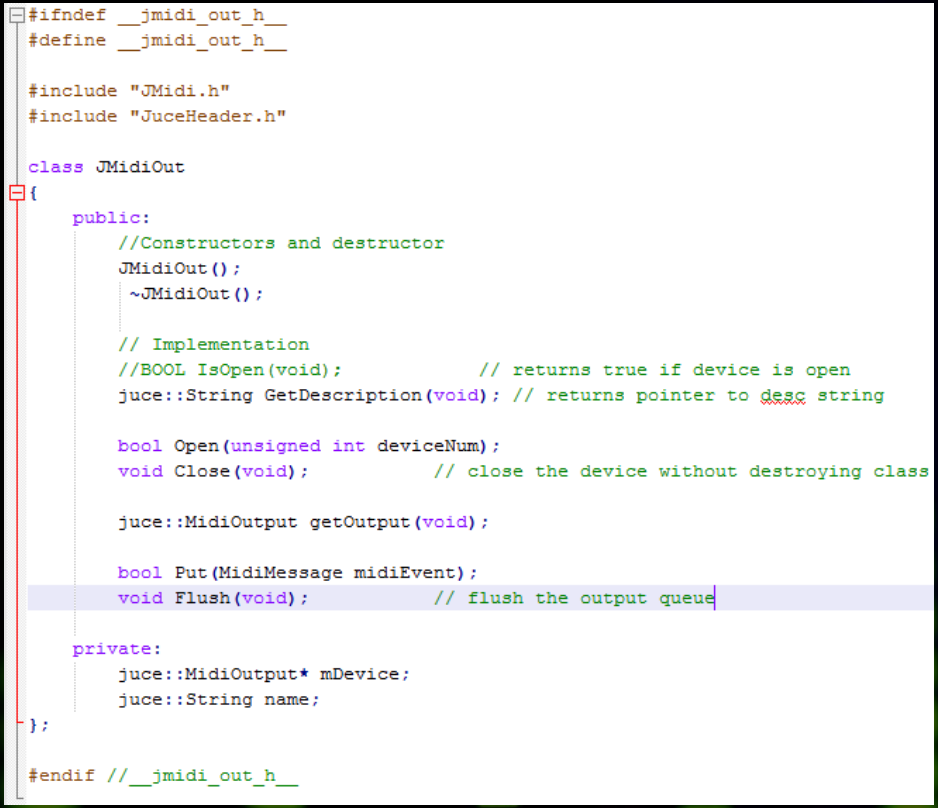
- Provide basic container for MIDI input device.
- Start and stop input devices from listening to ports.
- MIDI message handling is done within class



```
#ifndef __jmidi_in_h__
#define __jmidi_in_h__

#include "JMidi.h"
#include "JuceHeader.h"

class JMidiIn
{
    public:
        //Constructors and destructor
        JMidiIn();
        ~JMidiIn();

        // Implementation
        //BOOL IsOpen(void);            // returns true if device is open
        juce::String GetDescription(void);  // returns pointer to desc string

        BOOL Open(UINT deviceNum);
        void Close(void);               // close the device without destroying class

        void Start(void);               // start midi in
        void Stop(void);                // stop midi in

        juce::MidiInput getInput(void);

        void handleIncomingMidiMessage(juce::MidiInput*,const juce::MidiMessage&);

    private:
        juce::MidiInput* mDevice;
        juce::String name;
};

#endif //__jmidi_in_h__
```

JMidiOut

- Provide basic container for MIDI output device.
- Simple for other classes to open and close MIDI output devices.
- Pushing MIDI to output ports handled internally



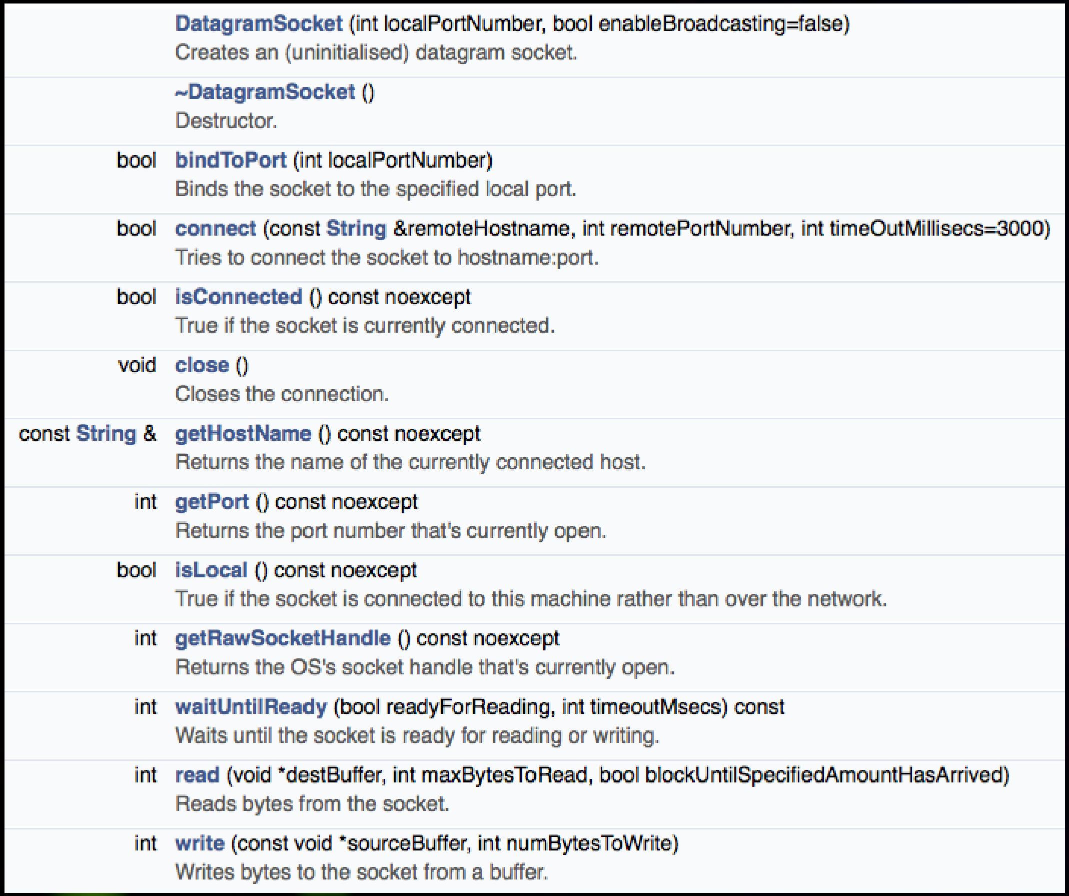
```
#ifndef __jmidi_out_h__
#define __jmidi_out_h__

#include "JMidi.h"
#include "JuceHeader.h"

class JMidiOut
{
    public:
        //Constructors and destructor
        JMidiOut();
         ~JMidiOut();

        // Implementation
        //BOOL IsOpen(void);             // returns true if device is open
        juce::String GetDescription(void); // returns pointer to desc string

        bool Open(unsigned int deviceNum);
        void Close(void);            // close the device without destroying class

        juce::MidiOutput getOutput(void);

        bool Put(MidiMessage midiEvent);
        void Flush(void);            // flush the output queue

    private:
        juce::MidiOutput* mDevice;
        juce::String name;
};

#endif //__jmidi_out_h__
```

MIDINetworking

- Original MIDINet used multicast messaging.
- JUCE does not provide support for joining multicast groups.
- Multicast only used for group messaging.
- For sake of cross-platform ease: Broadcast
- Using JUCE’s datagram sockets

Datagram Sockets

- Provides all the necessary method for sending and receiving UDP messages.
- A few issues…



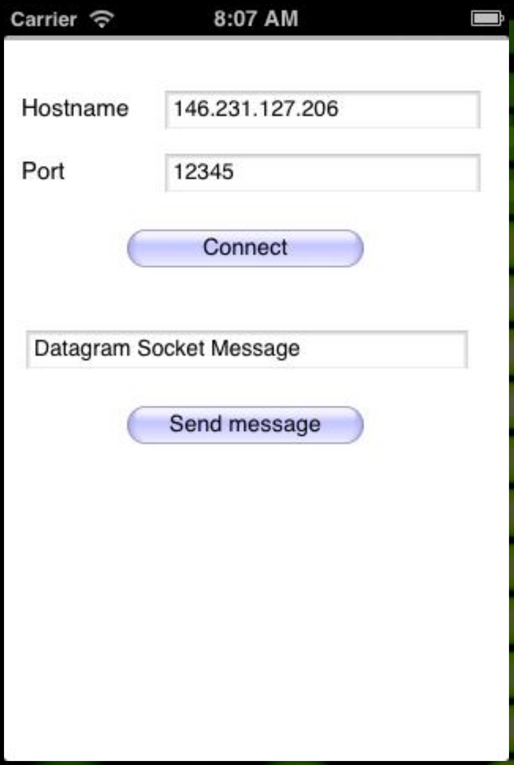
	DatagramSocket (int localPortNumber, bool enableBroadcasting=false) Creates an (uninitialised) datagram socket.
	~DatagramSocket () Destructor.
bool	bindToPort (int localPortNumber) Binds the socket to the specified local port.
bool	connect (const String &remoteHostname, int remotePortNumber, int timeOutMillisecs=3000) Tries to connect the socket to hostname:port.
bool	isConnected () const noexcept True if the socket is currently connected.
void	close () Closes the connection.
const String &	getHostName () const noexcept Returns the name of the currently connected host.
int	getPort () const noexcept Returns the port number that's currently open.
bool	isLocal () const noexcept True if the socket is connected to this machine rather than over the network.
int	getRawSocketHandle () const noexcept Returns the OS's socket handle that's currently open.
int	waitUntilReady (bool readyForReading, int timeoutMsecs) const Waits until the socket is ready for reading or writing.
int	read (void *destBuffer, int maxBytesToRead, bool blockUntilSpecifiedAmountHasArrived) Reads bytes from the socket.
int	write (const void *sourceBuffer, int numBytesToWrite) Writes bytes to the socket from a buffer.

Networking Fixes

- Host IPAddress identifier.
- MIDINet packet.
  - Contains both a sender's IPAddress as well as MIDINet message.
  - Alternative: Fix JUCE DatagramSocket class to allow return of sender IP

Datagram Sockets

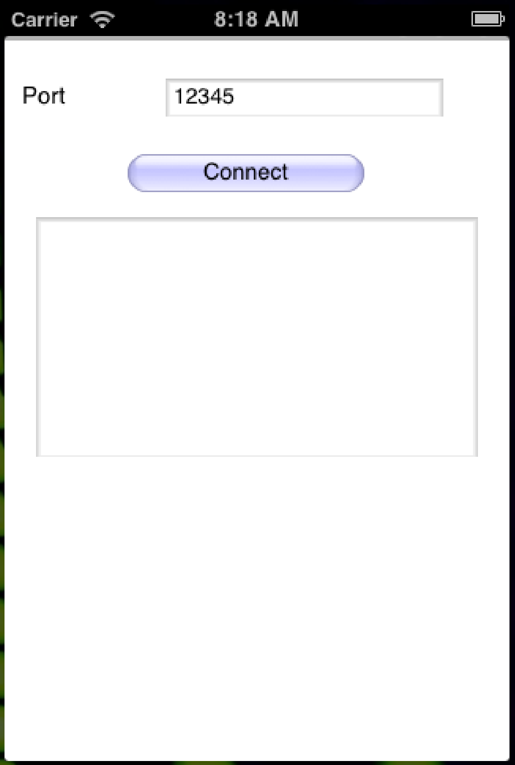
- Simple UDP messenger using JUCEDatagramSockets.
- Tested on Linux, Windows, OSX and iOS on both wired and wireless networks.



Hostname 146.231.127.206

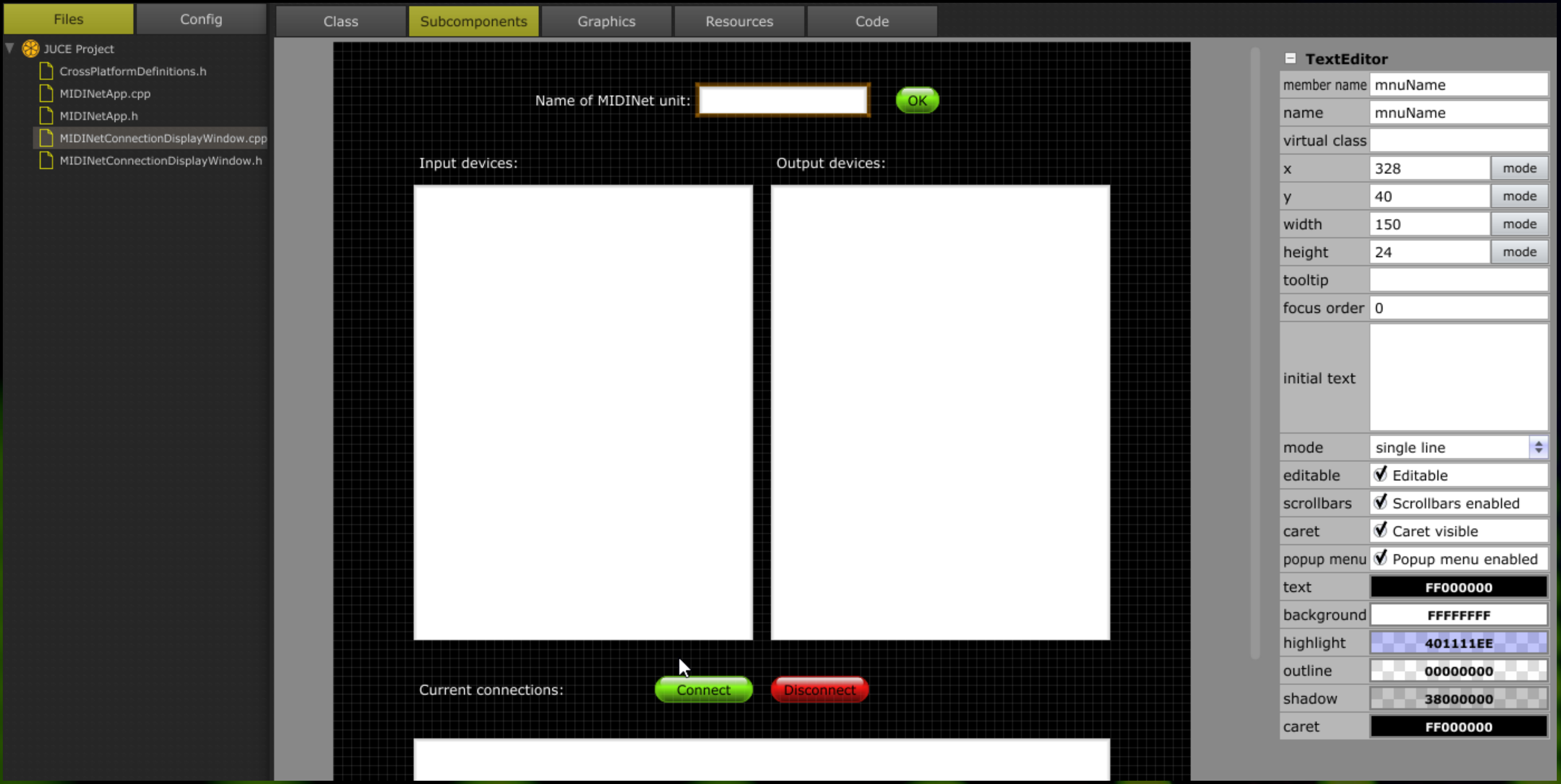
Port 12345

Datagram Socket Message



Port 12345

New user interface

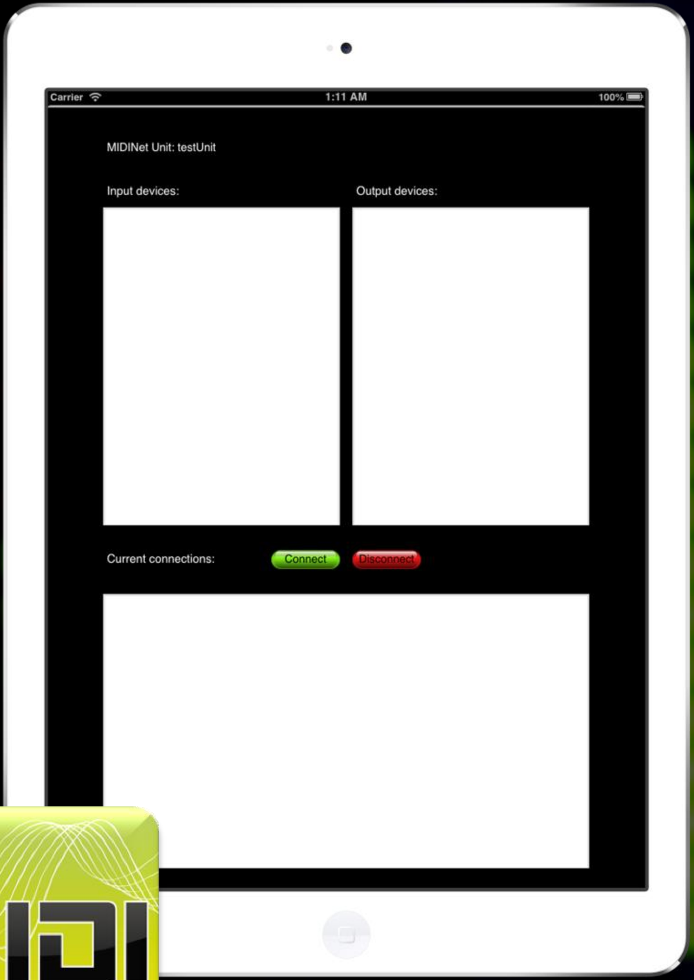


TextEditor

member name	mnuName	
name	mnuName	
virtual class		
x	328	mode
y	40	mode
width	150	mode
height	24	mode
tooltip		
focus order	0	
initial text		
mode	single line	
editable	✓ Editable	
scrollbars	✓ Scrollbars enabled	
caret	✓ Caret visible	
popup menu	✓ Popup menu enabled	
text	FF000000	
background	FFFFFFFF	
highlight	401111EE	
outline	00000000	
shadow	38000000	
caret	FF000000	

Using the UI

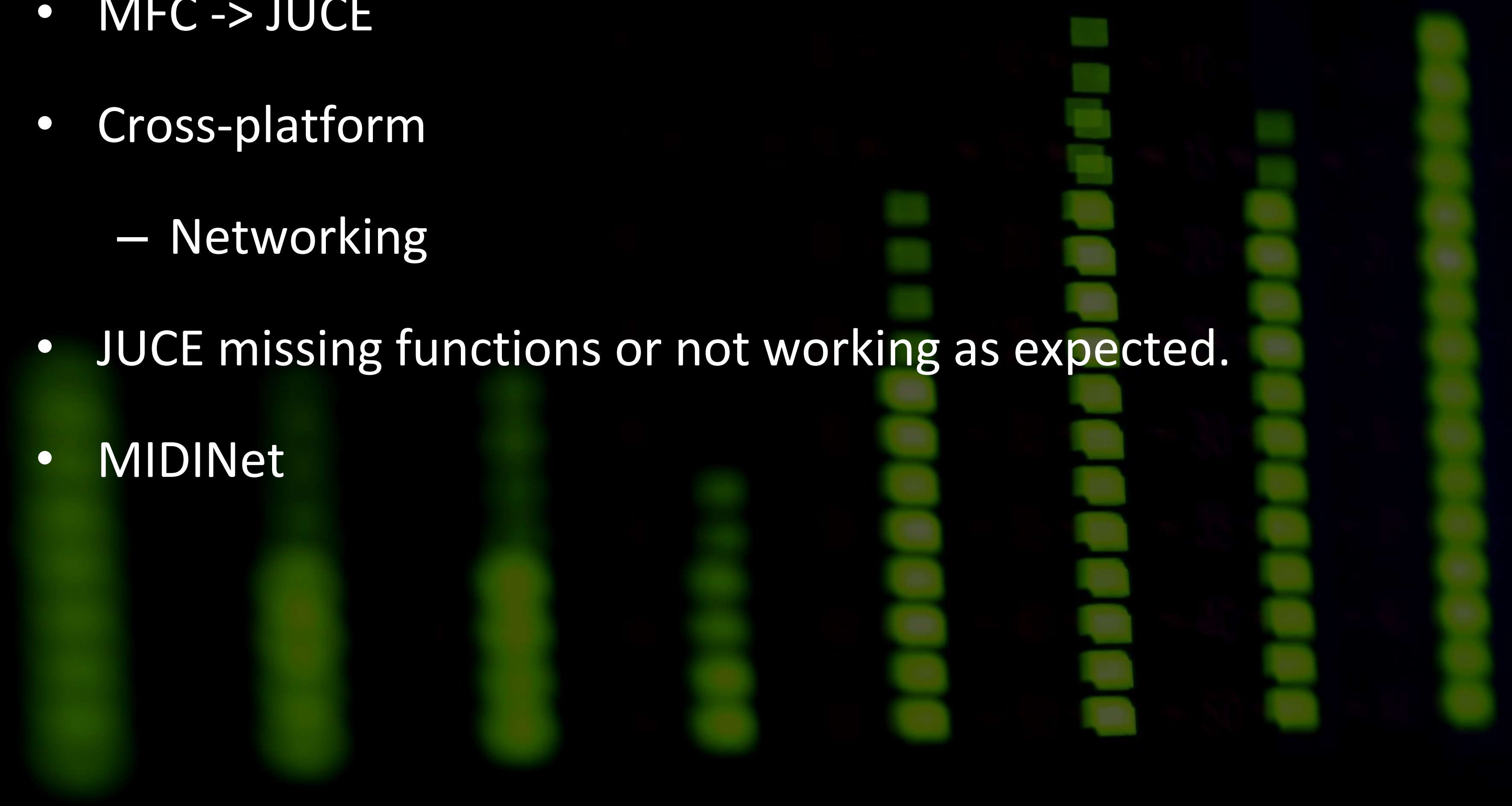
- User interface is now a smooth, single screen touch application.
- Input and output devices will be selected from the ListBoxes on the left and right.
- Current connections will be seen in the box below.



MFC to JUCE/standard libraries

- CString -> String
- CObject
- CTypedPtrList -> List<Type> Name
  - Iterators
- AfxMessageBox
- A lot of small changes
  - UINT

Issues



- MFC -> JUCE
- Cross-platform
  - Networking
- JUCE missing functions or not working as expected.
- MIDINet

Summary

- JUCE
- MIDINet on the iPad
- Future directions