```
//
//  USBSerial.swift
//  RGB USB
//
//  Created by Erik Nordlund on 4/30/19.
//  Copyright © 2019 Erik Nordlund. All rights reserved.
//
//  Arm USB includes the following open-source components:
//      • peertalk-simple: https://github.com/kirankunigiri/peertalk-simple
//      • ORSSerialPort: https://github.com/armadsen/ORSSerialPort

import Foundation

var usbSerial: USBSerial!

protocol USBSerialDelegate {
    func messageWasReceived(_ message: String)

}

protocol USBSerialPortDelegate {
    func portDidSet()
}


class USBSerial : NSObject {

    var delegate: USBSerialDelegate!

    var portDelegate: USBSerialPortDelegate!

    init(delegate: USBSerialDelegate) {
        debugPrint("USBSerial init()")
        super.init()
        self.delegate = delegate

        // Initialize serial
        debugPrint(self.serialPort)
        self.serialPort = ORSSerialPort(path: "/dev/cu.usbmodem14201")
        debugPrint(self.serialPort)
    }

    let serialPortManager = ORSSerialPortManager.shared()

    //var currentState = ApplicationState.initializationState
    let standardInputFileHandle = FileHandle.standardInput
    //let prompter = UserPrompter()

    var serialPort: ORSSerialPort? {
        didSet {
            debugPrint("didSet serialPort")
```

```
        oldValue?.close()
        oldValue?.delegate = nil

        self.serialPort?.baudRate = 9600
        self.serialPort?.delegate = self;
        self.serialPort?.open()

        portDelegate.portDidSet()
    }
}


var incomingBuffer: String? {
    didSet {
        if let message = self.incomingBuffer {
            let lastCharacter = message.last
            switch lastCharacter {
            case ".":
                self.incomingBuffer = nil
                delegate.messageWasReceived(message)
                debugPrint("message received: ", message)
            case "!":
                self.incomingBuffer = nil
                delegate.messageWasReceived(message)
                debugPrint("message received: ", message)
            case "?":
                self.incomingBuffer = nil
                delegate.messageWasReceived(message)
                debugPrint("message received: ", message)
            default:
                return
            }
        }
    }
}

func toggleConnection() {
    if self.serialPort != nil {
        self.serialPort = nil
    } else {
        self.serialPort = ORSSerialPort(path: "/dev/cu.usbmodem14201")
    }

    /*
    if let port = self.serialPort {
        if (port.isOpen) {
            debugPrint("port is open. closing port.")
            port.close()
        } else {
            debugPrint("port is closed. opening port.")
            port.open()
```

```
            }
        }
*/
    }


    func runProcessingInput() {
        setbuf(stdout, nil)
        standardInputFileHandle.readabilityHandler = { (fileHandle:
         FileHandle) in
            let data = fileHandle.availableData
            DispatchQueue.main.async {
                self.handleUserInput(data)
            }
        }

        //prompter.printIntroduction()

        let availablePorts = ORSSerialPortManager.shared().availablePorts
        if availablePorts.count == 0 {
            print("No connected serial ports found. Please connect your USB to
             serial adapter(s) and run the program again.\n")
            exit(EXIT_SUCCESS)
        }
        //prompter.promptForSerialPort()
        //currentState = .waitingForPortSelectionState(availablePorts)

        RunLoop.current.run() // Required to receive data from ORSSerialPort
         and to process user input
    }

    // MARK: Port Settings
    func setupAndOpenPortWithSelectionString(_ selectionString: String,
     availablePorts: [ORSSerialPort]) -> Bool {
        var selectionString = selectionString
        selectionString = selectionString.trimmingCharacters(in:
         CharacterSet.whitespacesAndNewlines)
        if let index = Int(selectionString) {
            let clampedIndex = min(max(index, 0), availablePorts.count-1)
            self.serialPort = availablePorts[clampedIndex]
            return true
        } else {
            return false
        }
    }

    func setBaudRateOnPortWithString(_ selectionString: String) -> Bool {
        var selectionString = selectionString
        selectionString = selectionString.trimmingCharacters(in:
         CharacterSet.whitespacesAndNewlines)
        if let baudRate = Int(selectionString) {
```

```
        self.serialPort?.baudRate = NSNumber(value: baudRate)
        print("Baud rate set to \(baudRate)", terminator: "")
        return true
    } else {
        return false
    }
}

// MARK: Data Processing
func handleUserInput(_ dataFromUser: Data) {
    if let nsString = NSString(data: dataFromUser, encoding:
     String.Encoding.utf8.rawValue) {

        let string = nsString as String

        if string.lowercased().hasPrefix("exit") ||
            string.lowercased().hasPrefix("quit") {
            print("Quitting...")
            exit(EXIT_SUCCESS)
        }
        /*
        switch self.currentState {
        case .waitingForPortSelectionState(let availablePorts):
            if !setupAndOpenPortWithSelectionString(string,
             availablePorts: availablePorts) {
                print("\nError: Invalid port selection.", terminator: "")
                prompter.promptForSerialPort()
                return
            }
        case .waitingForBaudRateInputState:
            if !setBaudRateOnPortWithString(string) {
                print("\nError: Invalid baud rate. Baud rate should
                 consist only of numeric digits.", terminator: "")
                prompter.promptForBaudRate();
                return;
            }
            currentState = .waitingForUserInputState
            prompter.printPrompt()
        case .waitingForUserInputState:
            self.serialPort?.send(dataFromUser)
            prompter.printPrompt()
        default:
            break;
        }
        */
    }
}

func send(message: String) {
    debugPrint("Trying to send message: ", message)
    if let data = message.data(using: String.Encoding.utf8) {
```

```
            self.serialPort?.send(data)
            debugPrint("message sent: ", message)
        }


    }
}


extension USBSerial: ORSSerialPortDelegate {
    func serialPort(_ serialPort: ORSSerialPort, didReceive data: Data) {
        if let string = String(data: data, encoding: String.Encoding.utf8) {
            //print("\nReceived: \"\(string)\" \(data)", terminator: "")
            if incomingBuffer != nil {
                incomingBuffer?.append(string)
            } else {
                incomingBuffer = string
            }
        }
        //prompter.printPrompt()
    }

    func serialPortWasRemoved(fromSystem serialPort: ORSSerialPort) {
        debugPrint("serialPortWasRemoved")
        self.serialPort = nil
    }

    func serialPort(_ serialPort: ORSSerialPort, didEncounterError error:
     Error) {
        print("Serial port (\(serialPort)) encountered error: \(error)")
    }

    func serialPortWasOpened(_ serialPort: ORSSerialPort) {
        print("Serial port \(serialPort) was opened", terminator: "")
        //prompter.promptForBaudRate()
        //currentState = .waitingForBaudRateInputState
    }
}
```