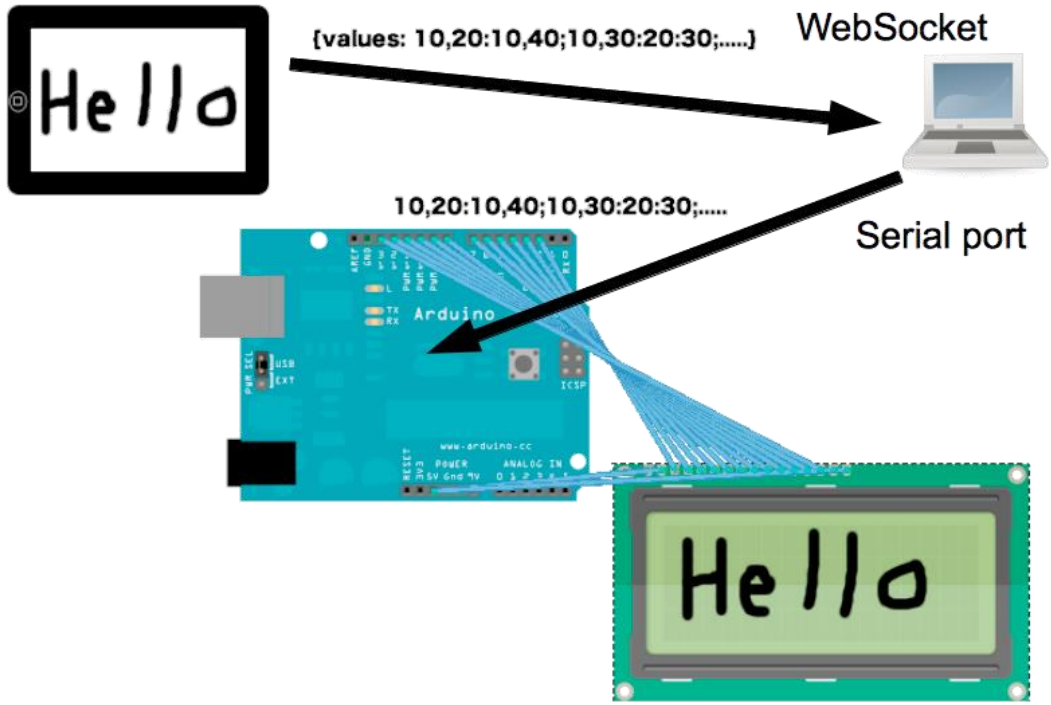
NodeJS and Serial Data

Professor:
Bladimir Bacca Cortes Ph.D.
Bladimir.bacca@correounivalle.edu.co

Grupo de Investigación en Percepción y Sistemas Inteligentes.

Contents

- Serialport package installation.
- Proteus test platform installation.
- API description.
- Serial port events.
- Listing serial ports in the system.
- Reading data from serial port
- Writing data from serial port
- Parsers on serial port.

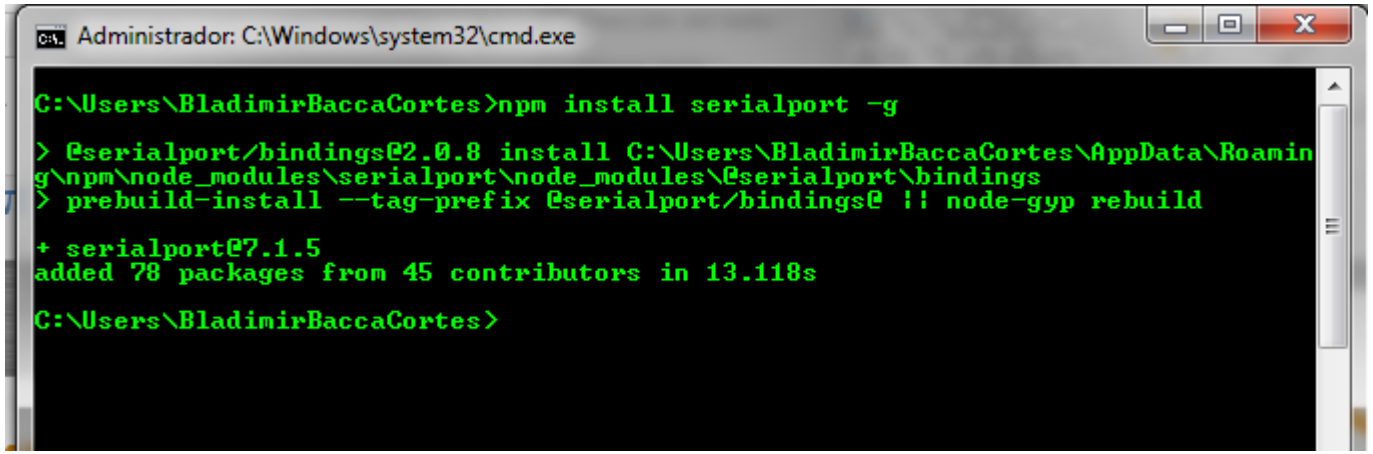


Serialport Package Installation

- Serialport library needs to be installed using npm.

Prompt> npm install serialport

- **Remember**:
  - If you install a module globally, it will not be able to be imported by require().
  - -g parameter defines a global installation.
  - Otherwise, in the folder application will appear a subdirectory called **node_modules**, and in that directory it will install all the necessary assets for the **serialport** library.
  - Docs: https://serialport.io/docs/10.x.x/

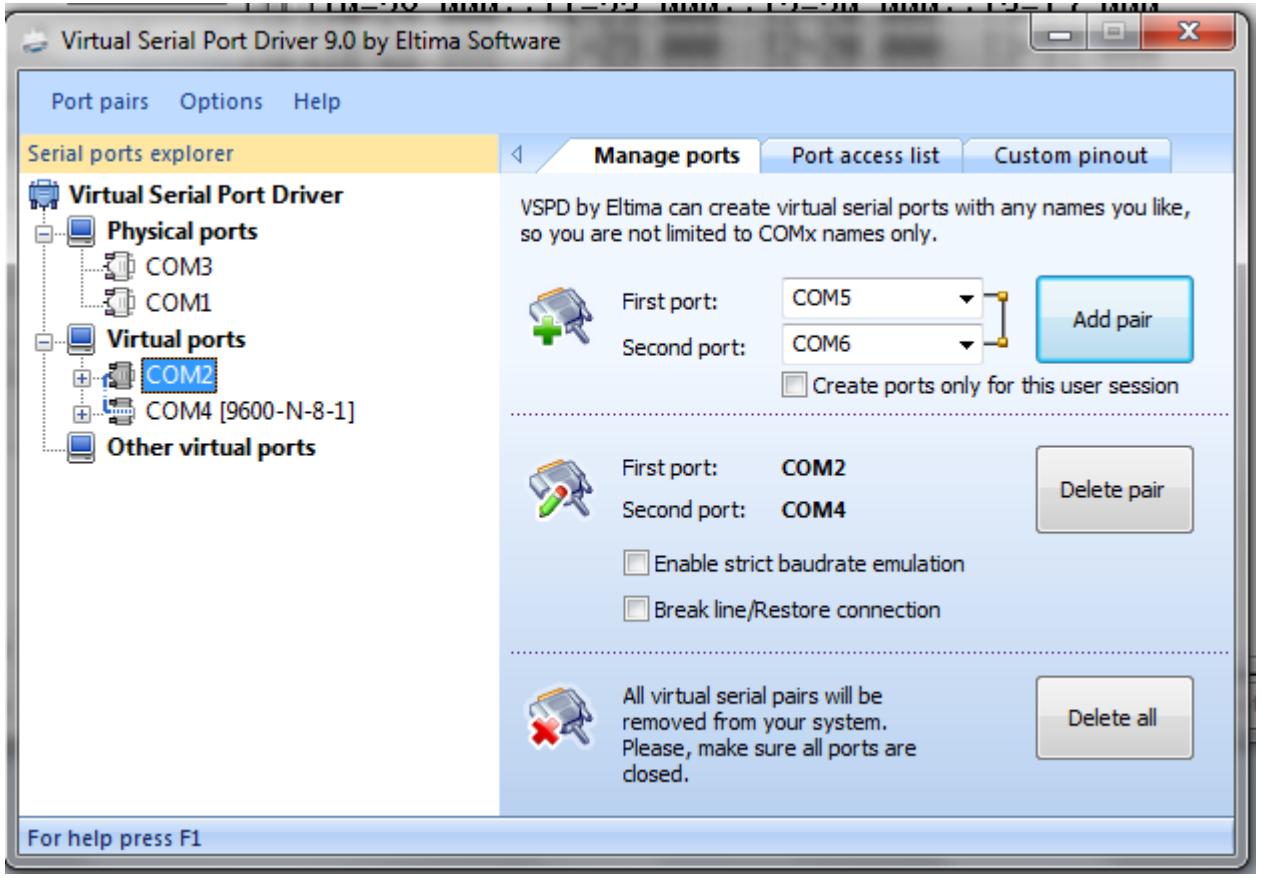


Proteus Test Platform Installation

- **Problem**: Lot of laptops do not have serial ports, and for testing point of view is more comfortable not using hardware attached to serial ports.

- **Solution**: Serial Terminal – Proteus + COMPIM + Virtual serial ports.

- **Basic steps – Virtual serial ports**:
  - Go to https://www.virtual-serial-port.org/ and download the Virtual Serial Port Driver
  - Execute it.
  - Add a pair of serial ports.

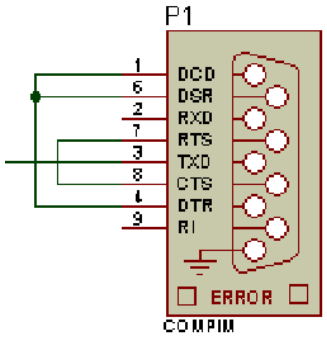


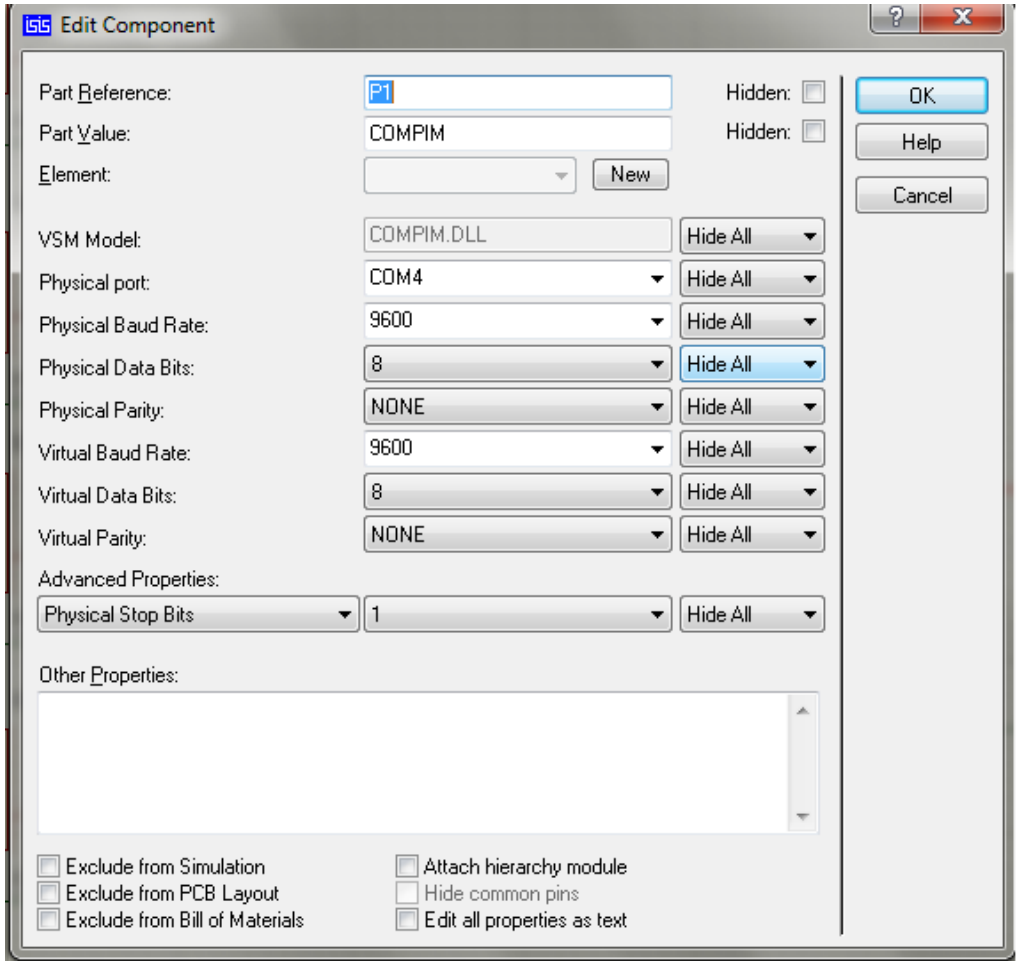
Proteus Test Platform Installation

- **Problem**: Lot of laptops do not have serial ports, and for testing point of view is more comfortable not using hardware attached to serial ports.

- **Solution**: Serial Terminal – Proteus + COMPIM + Virtual serial ports.

- **Basic steps – COMPIM**:
  - Load the **nodeJSSerialPort** Proteus project.
  - Edit the COMPIM component.
  - Connect it to the COM4 using 9600-N-8-No parity configuration.



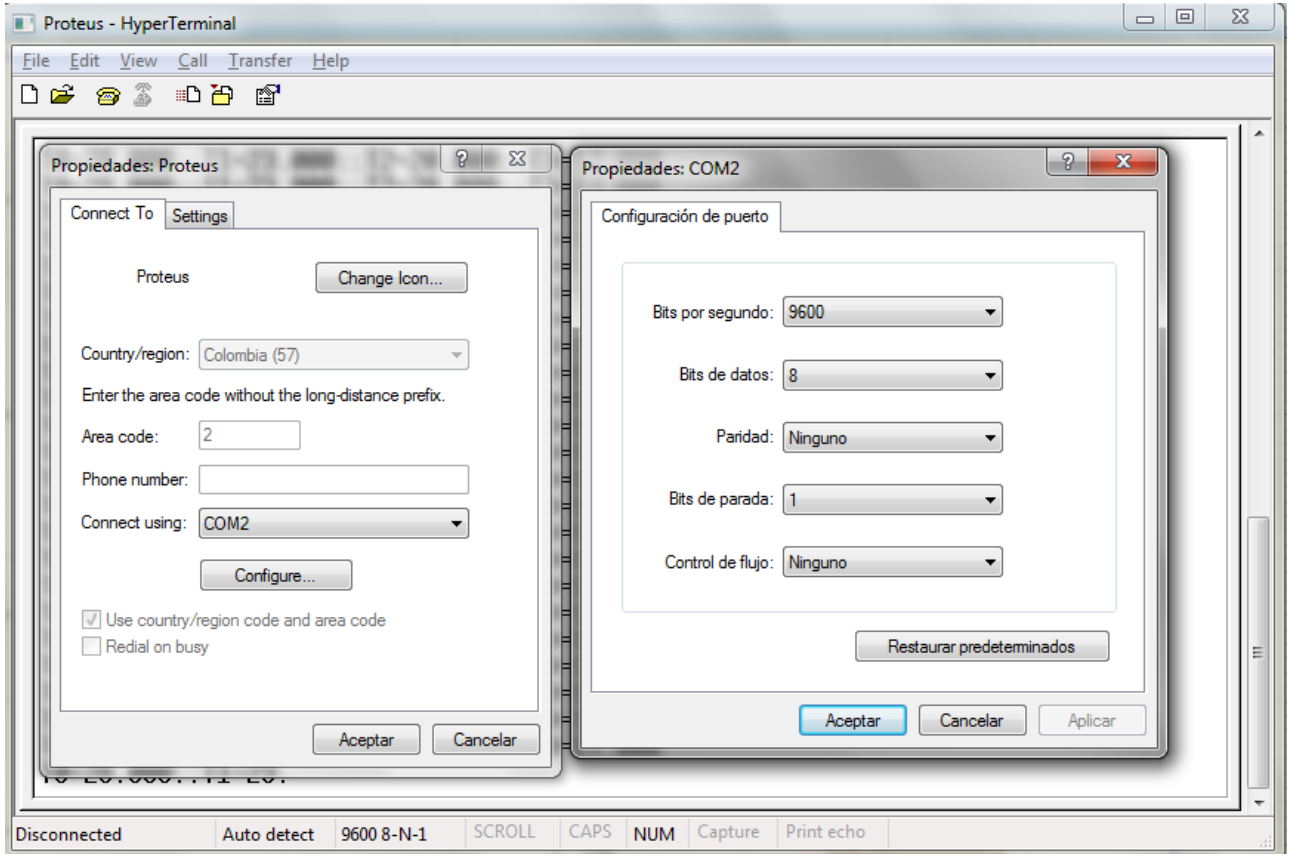


Proteus Test Platform Installation

- **Problem**: Lot of laptops do not have serial ports, and for testing point of view is more comfortable not using hardware attached to serial ports.

- **Solution**: Serial Terminal – Proteus + COMPIM + Virtual serial ports.

- **Basic steps – Hyperterminal**:
  - Download Hyperterminal or any other Terminal software to connect to serial ports.
  - Execute it.
  - Configure it in order to connect to the COM2 serial port using 9600-N-8-No parity parameters.

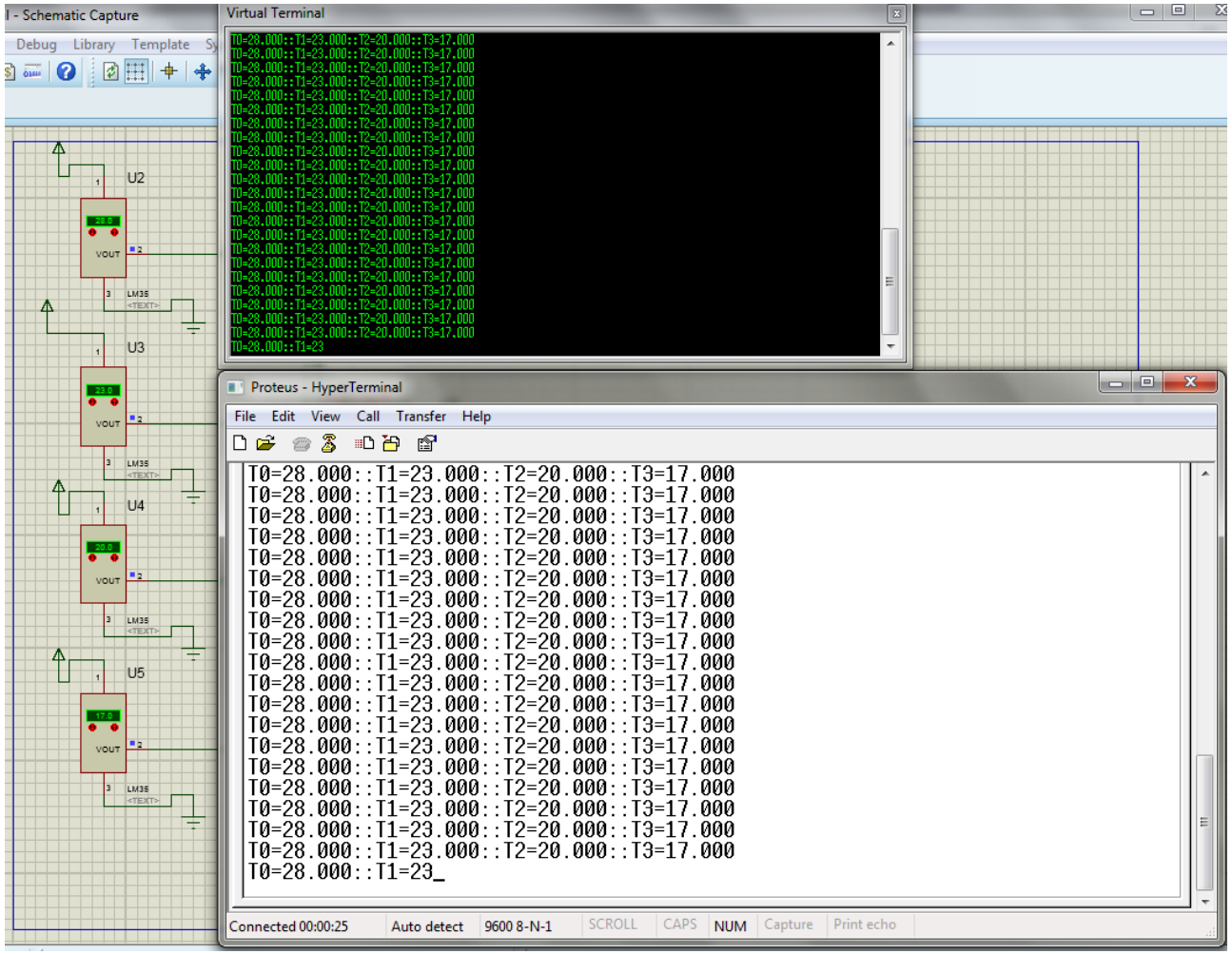


Proteus Test Platform Installation

- **Problem**: Lot of laptops do not have serial ports, and for testing point of view is more comfortable not using hardware attached to serial ports.

- **Solution**: Serial Terminal – Proteus + COMPIM + Virtual serial ports.

- **Final test**:
  - Load the nodeJSSerialPort Proteus project.
  - Execute it.
  - The temperature of 4 LM35 will be shown in the Proteus virtual terminal, and in the Hyperterminal software.



API Description

- **Stream interface**

const SerialPort = require('@serialport/stream')

const SerialPort = require('serialport')

- **Constructor and options**

new SerialPort(path [, openOptions] [, openCallback])

[autoOpen=true] Automatically opens the port on `nextTick`.

[baudRate=9600] The baud rate of the port to be opened.

[dataBits=8] Must be one of these: 8, 7, 6, or 5.

[highWaterMark=65536] The size of the read and write buffers.

[lock=true] Prevent other processes from opening the port.

[stopBits=1] Must be one of these: 1 or 2.

[parity=none] Must be one of these: 'none', 'even', 'mark', 'odd', 'space'.

[rtscts=false] flow control setting

[xon=false] flow control setting

[xoff=false] flow control setting

[xany=false] flow control setting

- **Serial port properties**

serialport.baudRate: number

serialport.isOpen: Boolean

serialport.path: string

serialport.binding: Binding

- **Events**
  - On open
  - On error
  - On close
  - On data
  - On drain (re-write information because of errors).

API Description

- **Standard methods**

serialport.open(() => {}): void

serialport.update(options: updateOptions, callback?: err => {}): void

serialport.write(data: string|Buffer|Array<number>, encoding?: string, callback?: error => {}): Boolean

serialport.read(size?: number): string|Buffer|null

serialport.close(callback?: error => {}): void

serialport.set(options: setOptions, callback?: error => {}): void

serialport.get(callback: (error, data: ModemStatus) => {}): void

serialport.flush(callback? error => {}):void

API Description

- **Standard Methods**

serialport.drain(callback? error => {}):void

serialport.pause(): this

serialport.resume(): this

- **Serial port Parsers**

  - **ByteLength**: Emit data every number of bytes.
  - **CCTalk**: A transform stream that emits ccTalk packets as they are received.
  - **Delimiter**: A transform stream that emits data each time a byte sequence is received. To use the Delimiter parser, provide a delimiter as a string, buffer, or array of bytes.
  - **Readline**: A transform stream that emits data after a newline delimiter is received. To use the Readline parser, provide a delimiter (defaults to \n).
  - **Ready**: A transform stream that waits for a sequence of "ready" bytes before emitting a ready event and emitting data events
  - **Regex**: A transform stream that uses a regular expression to split the incoming text upon.

Serial Port Events

- The **serialport** library, like most node.js libraries, is **event-based**.
- This means that when the program is running, the operating system and the user's actions will generate events and the program will provide functions to deal with those events called callback functions.

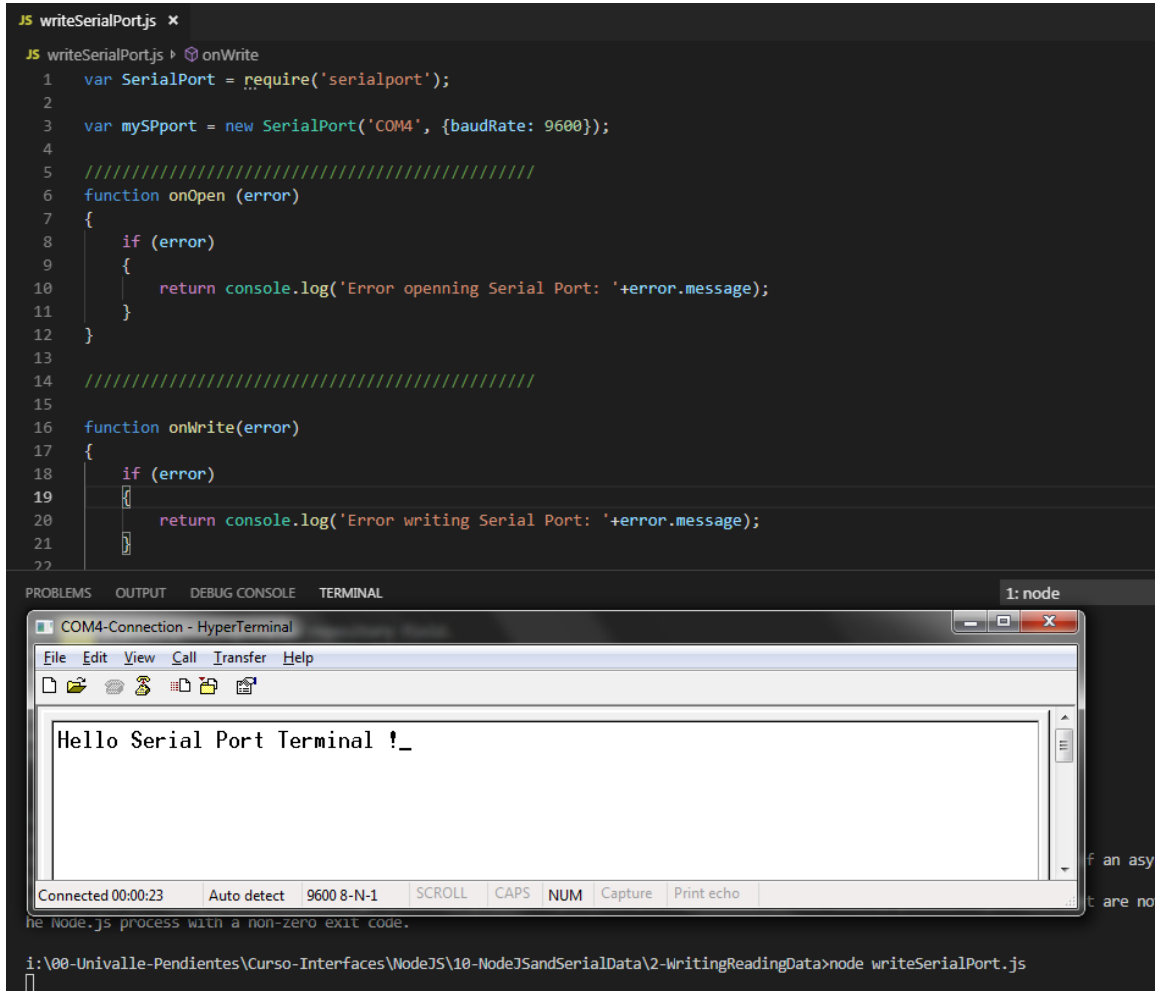
- **Events description**:
- Open: The open event happens when the port is opened and ready for writing. This happens if you have the constructor open immediately.
- Close: The close event's is emitted when the port is closed. In the case of a disconnect it will be called with a Disconnect Error object (err.disconnected == true).
- Error: The error provides an error object whenever there is an unhandled error. You can usually handle an error with a callback to the method that produced it.
- Data: Listening for the data event puts the port in flowing mode. Data is emitted as soon as it's received. Data is a Buffer object with any amount of data in it.
- Drain: The drain event is emitted when it is performant to write again if a write() call has returned false.

```
myPort.on('open', showPortOpen);
myPort.on('data', sendSerialData);
myPort.on('close', showPortClose);
myPort.on('error', showError);
```

```
function showPortOpen() {
  console.log('port open. Data rate: ' + myPort.options.baudRate); }
function sendSerialData(data) {
  console.log(data); }
function showPortClose() {
  console.log('port closed.'); }
function showError(error) {
  console.log('Serial port error: ' + error); }
```

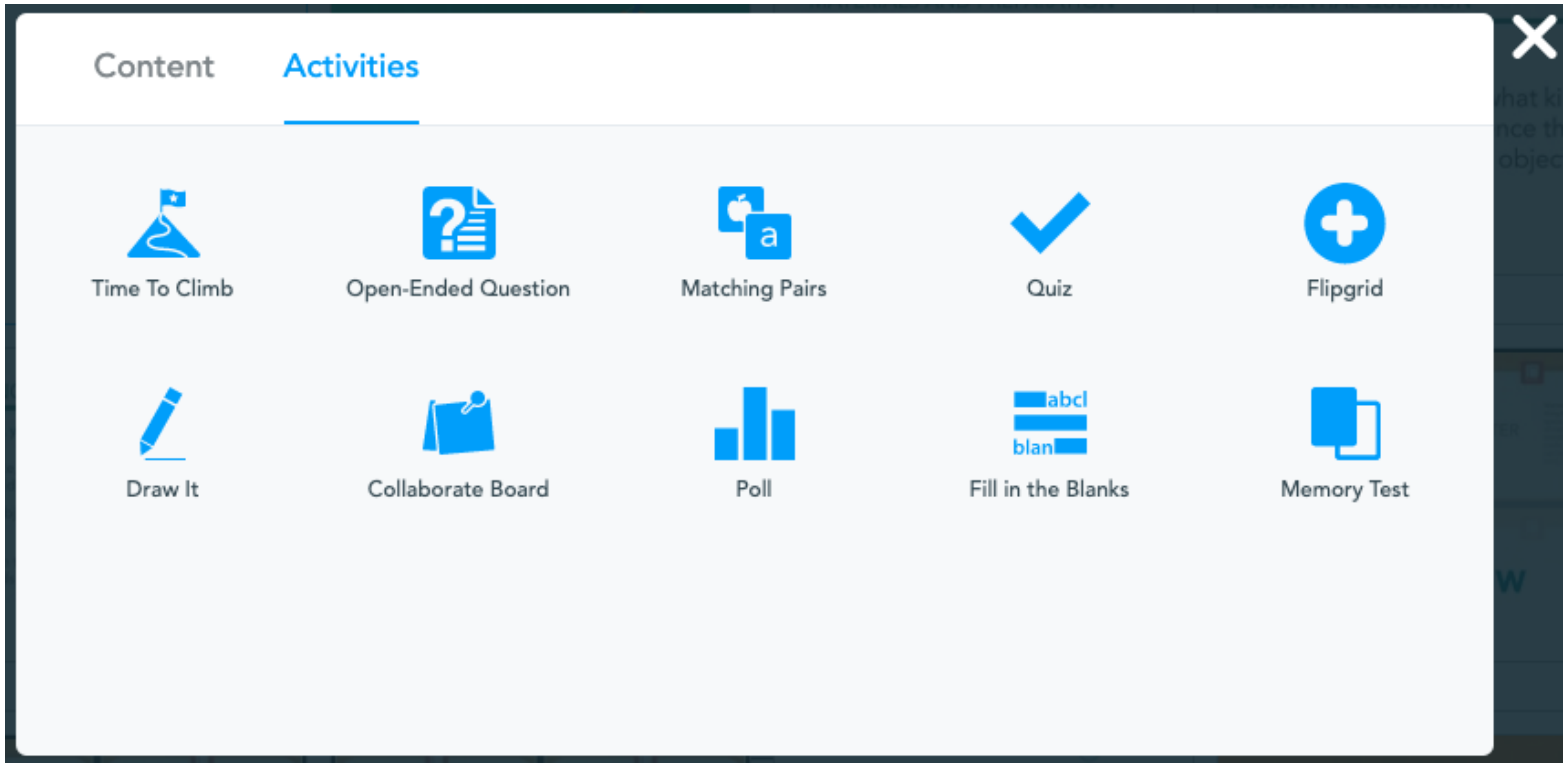
Writing Data to Serial Port

- The process of using the serialport library will be the same every time:
  - Initialize the serialport library
  - open the serial port
  - Set up the callback functions and let them do the rest
- **Remember**:
  - The properties such as stop bits, parity, data size have default values as follows:
  - *Stop bits: 1*
  - *Parity: none*
  - *Data size: 8*
  - You must configure the communication speed.
- **Note**: The Program Won't Stop!. When you run this program now, it won't automatically stop and return to the command line. To stop it, you'll need to type control-C in the terminal window to stop it.
- The new instance of Serialport created a software object that listens for events from the serial port. Any node.js script that creates an event listener like this will run until you explicitly stop it.
- Open folder **1-WrittingReadingData**, then run **writeSerialPort.js**.



Nearpod Activity

- Please go to the Nearpod link shared in the chat.
- Fulfil the Nearpod activity.
- Analyze the results with your teacher.



Parsers on Serial Port

- **Parsers**: Transform streams that process incoming data.
- To use the parsers, you must create them and then pipe the Serialport to the parser.
- Be careful to only write to the SerialPort object and not the parser.
- **Types**:

- ByteLength Parser: Emit data every number of bytes. A transform stream that emits data as a buffer after a specific number of bytes are received.
- Arguments: *options.length*: number the number of bytes to be emitted on each data event

```
new ByteLength(options)
```

- Delimiter Parser: A transform stream that emits data each time a byte sequence is received. To use the Delimiter parser, provide a delimiter as a string, buffer, or array of bytes

```
new Delimiter(options: { delimiter: string | Buffer | number[] })
```

- Arguments: *options.delimiter*: string|Buffer|number[] The delimiter in which to split incoming data.

- InterByteTimeout Parser: Emits data if there is a pause between packets for the specified amount of time. A transform stream that emits data as a buffer after not recieving any bytes for the specified amount of time.

```
new InterByteTimeout(options)
```

- Arguments: *options.interval*: number the period of silence in milliseconds after which data is emited.
- *options.maxBufferSize*: number the maximum number of bytes after which data will be emited. Defaults to 65536.

Parsers on Serial Port

- Readline Parser: A transform stream that emits data after a newline delimiter is received. To use the Readline parser, provide a delimiter (defaults to \n). Data is emitted as string controllable by the encoding option (defaults to utf8).

  new Readline(options?)

- Arguments: *options.delimiter?*: string delimiter to use
- *options.encoding?*: string text encoding for the stream

- Ready Parser: A transform stream that waits for a sequence of "ready" bytes before emitting a ready event and emitting data events. To use the Ready parser provide a byte start sequence.

  new Ready(options)

- Arguments: options.delimiter?: string delimiter to use to detect the input is ready

- Regex Parser: A transform stream that uses a regular expression to split the incoming text upon. To use the Regex parser provide a regular expression to split the incoming text upon.

  new Regex(options)

- Arguments: *options.regex:* RegExp the regular expression to use to split incoming text
- *options.encoding?:* string text encoding for the stream

- (Serial Line Internet Protocol) Slip Encoder Parser: A transform stream that emits SLIP-encoded data for each incoming packet.

  new SlipEncoder(options)

- Arguments: *options.bluetoothQuirk:* boolean Adds another 0xC0 character at the beginning if the bluetoothQuirk option is truthy

Reading Data from Serial Port

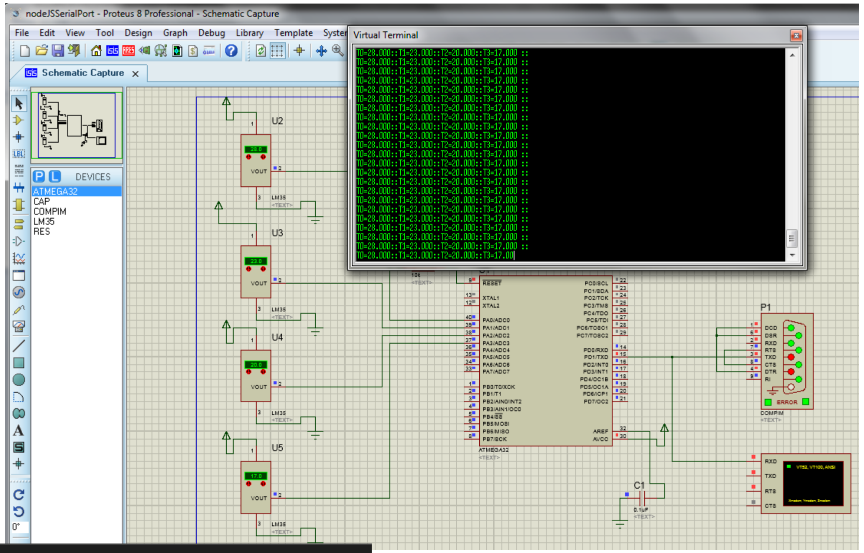
- Reading data from serial port requires using parsers for processing data:

```
mySerialPort.on('open', onOpen);

mySerialPort.open();

var myParserSP = mySerialPort.pipe(new ReadlineParser({delimiter: '\r\n'}));

myParserSP.on('data', function(data){
    console.log('Data received: ' + data);
});
```



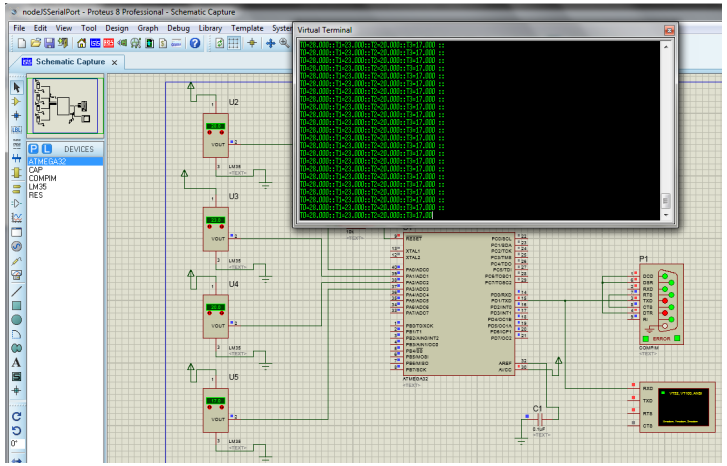
- Open **nodeJSSerialPort.pdsprj** Proteus project, then run it.
- Open 1-**WrittingReadingData**, then run **readSerialPort.js**.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
Data received: T0=28.000::T1=23.000::T2=20.000::T3=17.000 ::
```

Reading Raw Data from Serial Port

- Reading encapsulated data on serial port: Pattern detection.
- To do that, parsers must be used: **Delimiter parser**.
- Then, Buffers must be used to encode raw data.



```
var myParserSP = mySerialPort.pipe(new DelimiterParser({delimiter: '~~'}));

myParserSP.on('data', function(data){
    console.log('Data Frame Received: ', data);
    console.log('Temperature Ch0: ', data.readFloatLE(0));
    console.log('Temperature Ch1: ', data.readFloatLE(4));
    console.log('Temperature Ch2: ', data.readFloatLE(8));
    console.log('Temperature Ch3: ', data.readFloatLE(12));
```

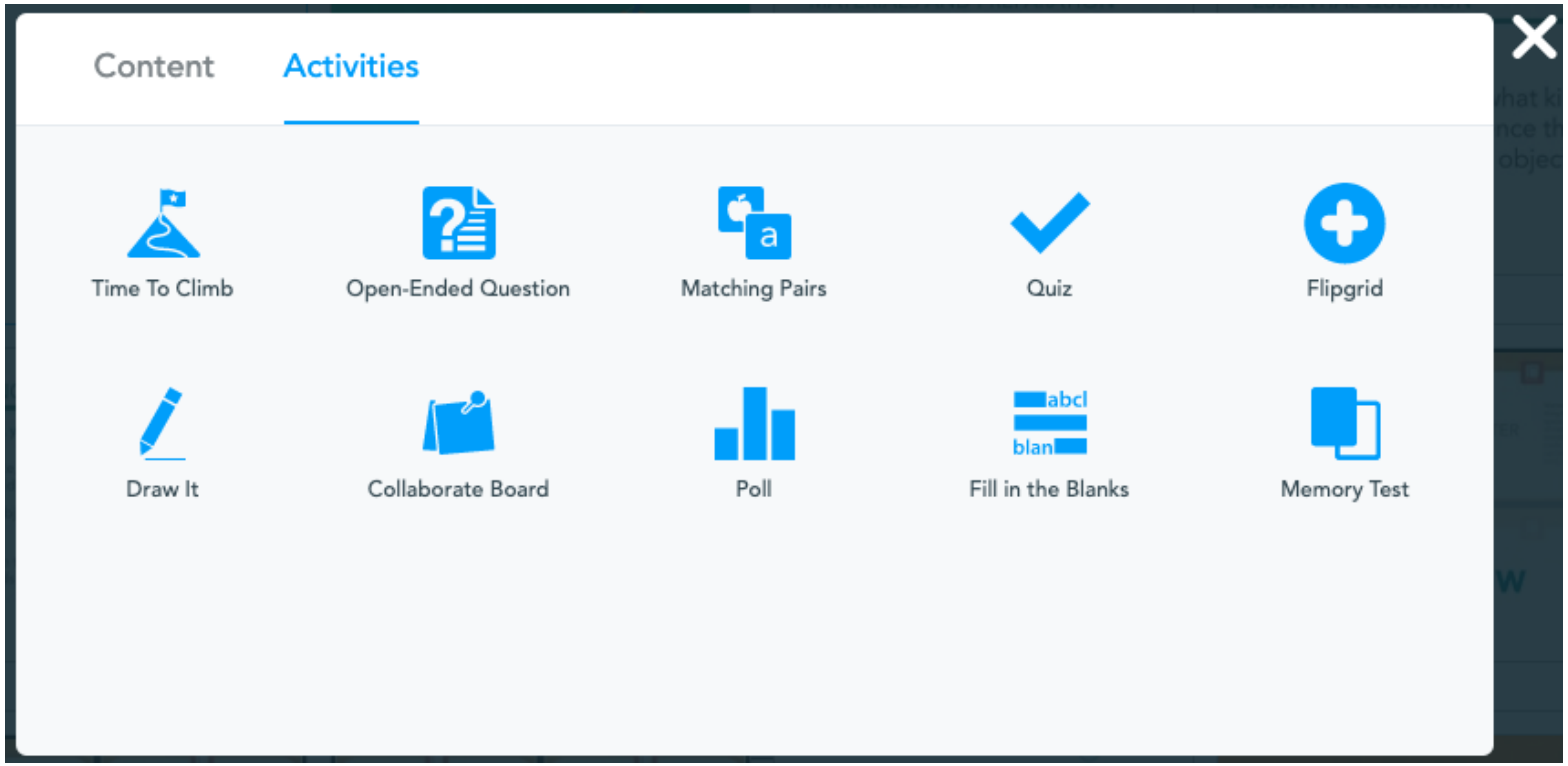
- Open **nodeJSSerialPort-V2.pdsprj** Proteus project, then run it.
- Open **1-WrittingReadingData**, then run **readFloatSerialPort.js**.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Temperature Ch3:  15
Data Frame Received:  <Buffer 00 00 04 42 00 00 a0 41 00 00 98 41 00 00 70 41>
Temperature Ch0:  33
Temperature Ch1:  20
Temperature Ch2:  19
Temperature Ch3:  15
Data Frame Received:  <Buffer 00 00 04 42 00 00 a0 41 00 00 98 41 00 00 70 41>
Temperature Ch0:  33
Temperature Ch1:  20
Temperature Ch2:  19
Temperature Ch3:  15
Data Frame Received:  <Buffer 00 00 04 42 00 00 a0 41 00 00 98 41 00 00 70 41>
Temperature Ch0:  33
Temperature Ch1:  20
Temperature Ch2:  19
Temperature Ch3:  15
```

Nearpod Activity

- Please go to the Nearpod link shared in the chat.
- Fulfil the Nearpod activity.
- Analyze the results with your teacher.



Questions?