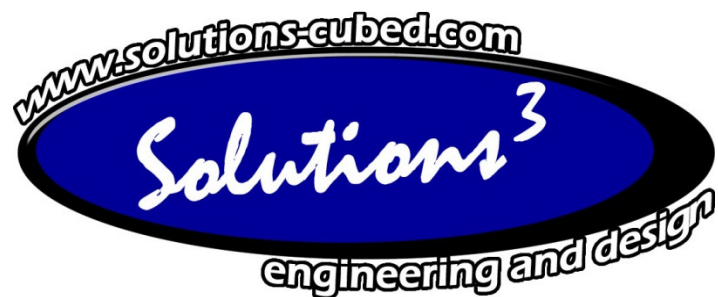


# **AN2011: Using the Synaptron Micro with serial control and encoder feedback.**



**Solutions Cubed, LLC**

**designservices@solutions-cubed.com**

**phone – 530.891.8045**

**256 East First Street**

**Chico, CA 95928**

**Contact Solutions Cubed, LLC for your custom designs:**

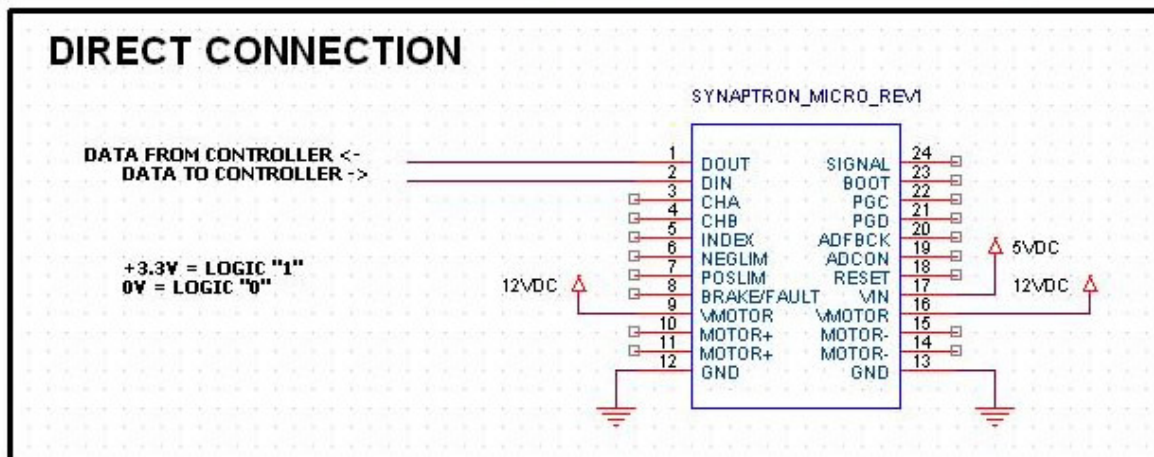
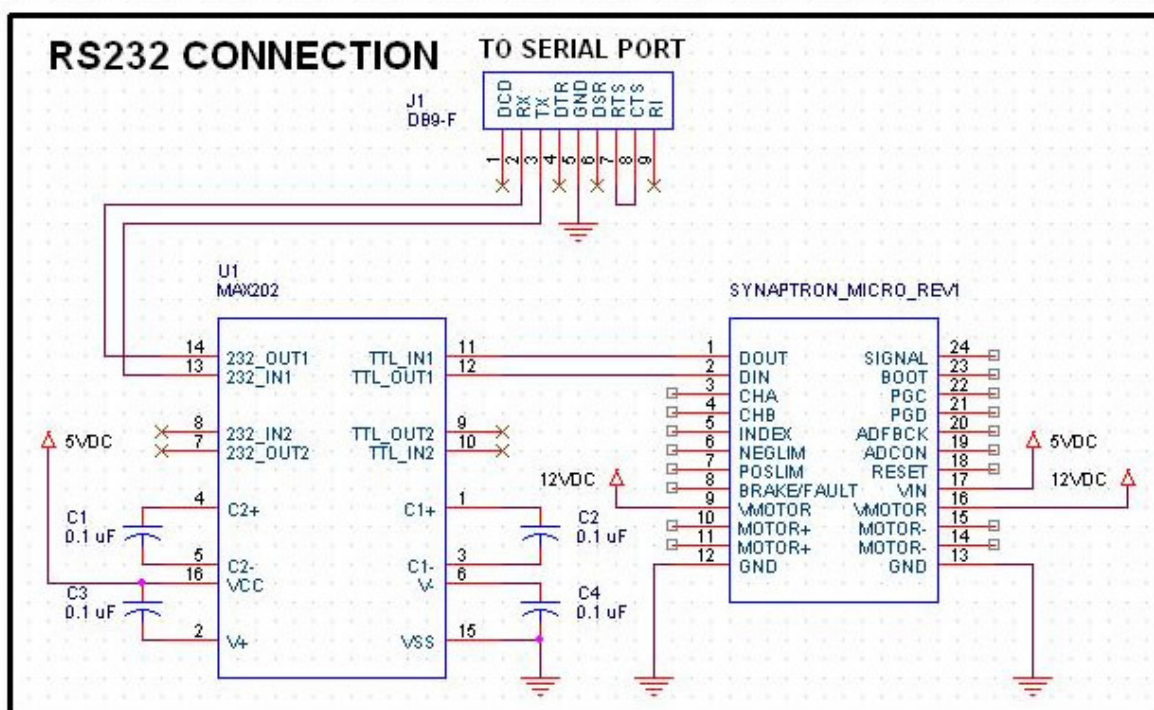
Solutions Cubed is an innovative electronic design firm. We have created successful designs for a myriad of industries including mass produced consumer products, deep-sea robotic components, and encrypted encoders for the banking industry. We love meeting new customers and are interested in hearing about your design needs.

## Overview-

This application note describes how to configure the Synaptron Micro to operate with serial data commands and quadrature encoder feedback in a position control system. The configuration will occur using our test software.

The Synaptron Micro accepts serial commands on its DIN pin (pin 2) and sends responses on its DOUT pin (pin 1). The DIN pin is 5V tolerant, but the DOUT pin will send a logic high signal at 3.3V. In this example we'll use an RS232 converter to allow the Synaptron to be programmed with our test software and a PC (upper schematic).

To accompany this application note you should consider downloading the user datasheet, communication protocol, and test software from [www.solutions-cubed.com](http://www.solutions-cubed.com).



## Encoder and Motor Connections-

The encoder and motor connections are fairly straightforward, and are shown in the schematic below. The INDEX connection is not necessary. All of the encoder inputs on the Synaptron Micro are 5V tolerant.

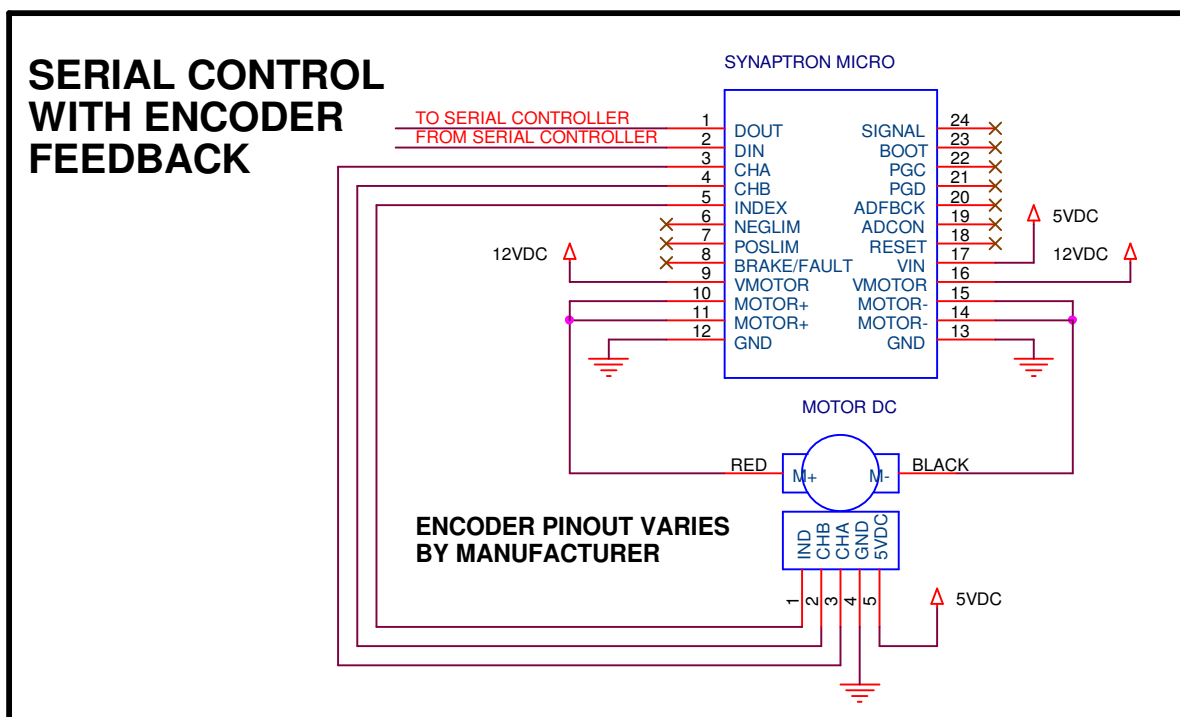
### Hints:

**1. 16-bit mode vs. 32-bit mode** – The controller defaults to 16-bit mode where the position value is stored in the register 5 and positions from -32,768 to 32,767 are counted. Set the *Function.F\_32Pos* bit to enable 32-bit position values.

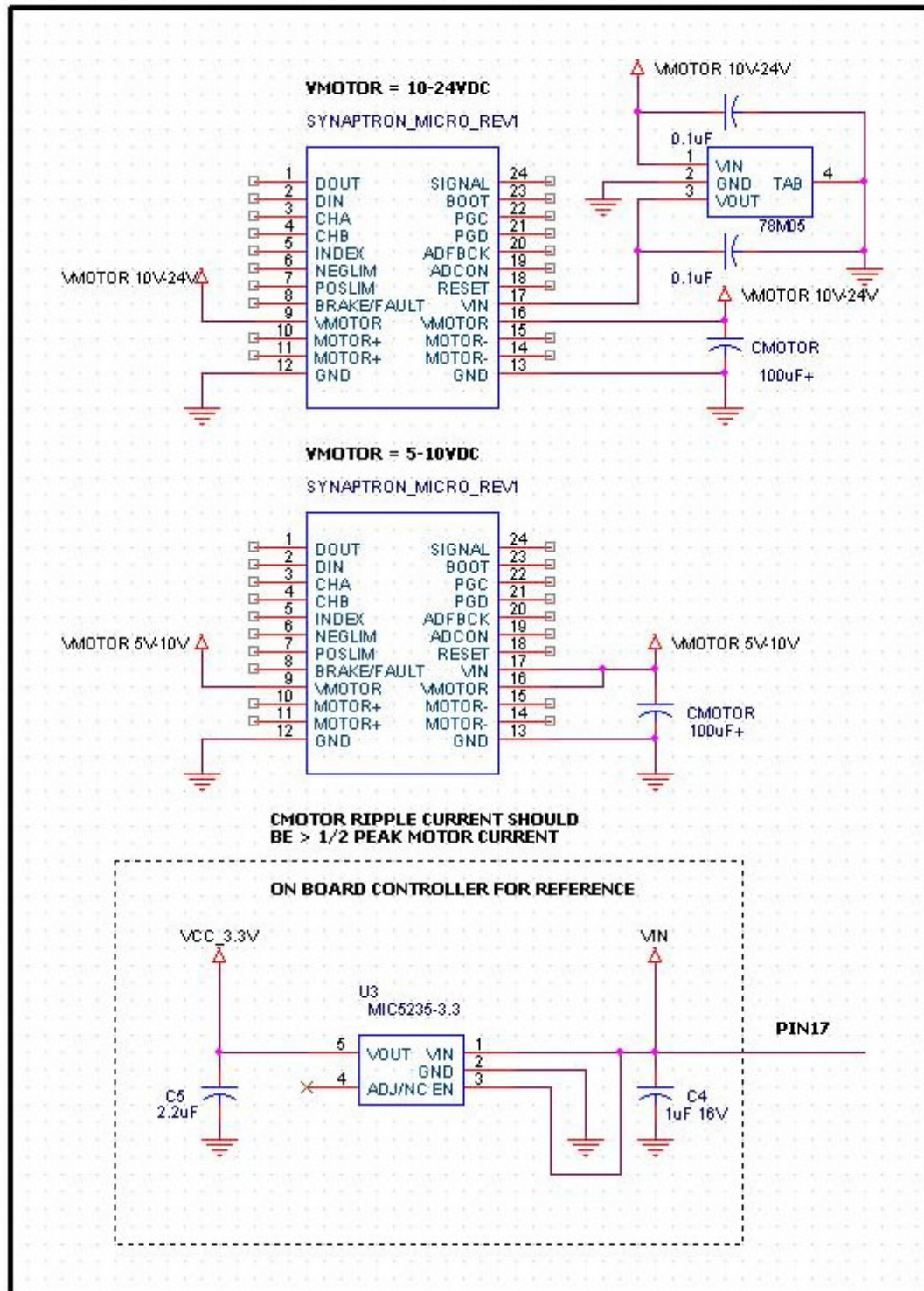
**2. Encoder CPR** – The controller creates a 2x count of encoder pulses. A 1000 count-per-revolution (1000CPR) encoder that moves to position 1000 will record a position of 2000 in the *PositionLow* register (#5).

**3. Swapping A-B** – If your system counts down when the motor moves forward (or vice-versa) you need to swap the A-B channels or reverse your motor connections. Setting the *Function.F\_SwapAB* bit will also reverse the direction of your count.

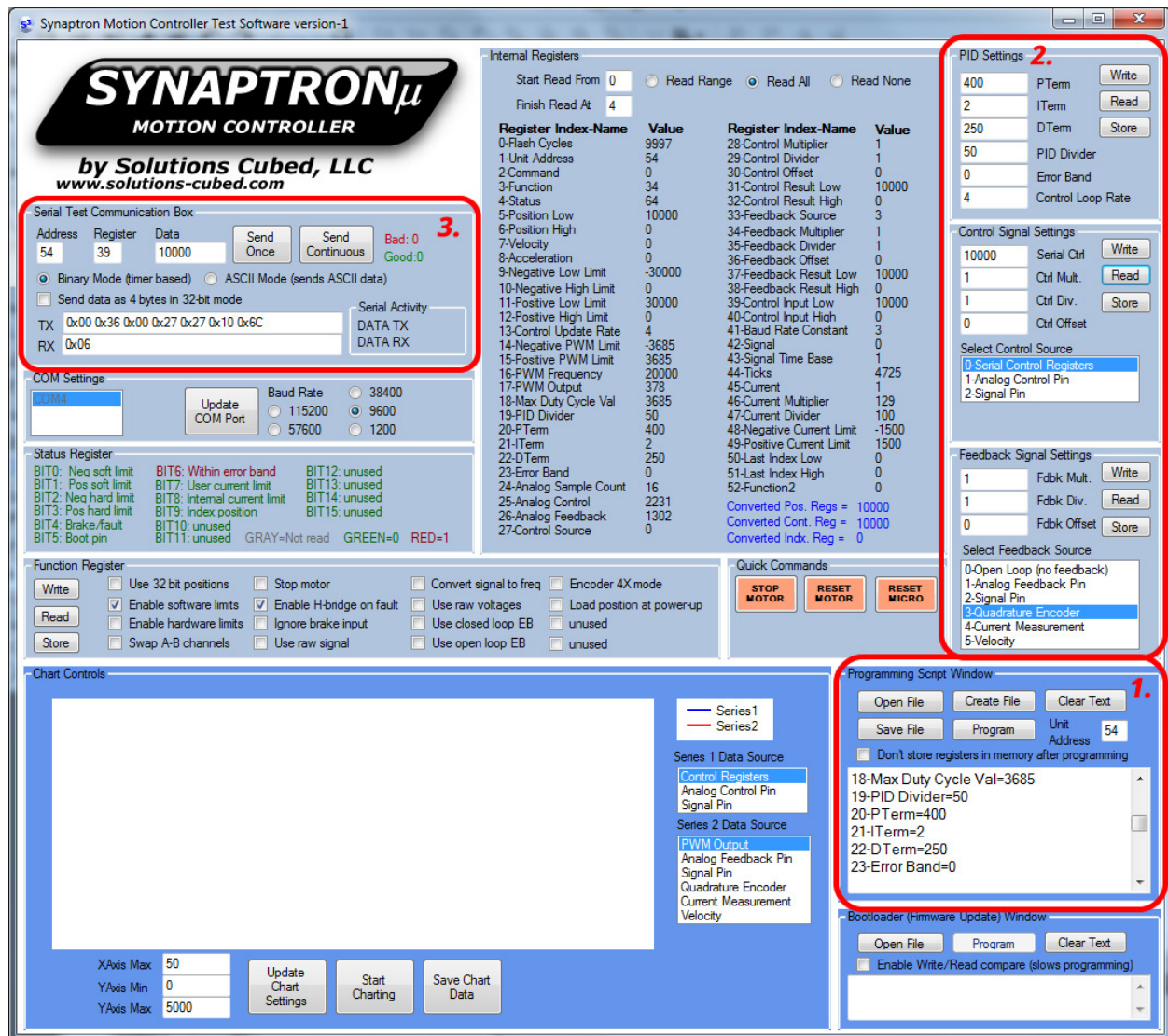
**4. Sending position commands:** Changes to various registers are described later. But in order to send a desired position value to the Synaptron you'll write the position to the *ControlInputLow* register (register 39). If operating in 32-bit mode you'll write a 32-bit value to *ControlInputHigh* (register 40) and the lower 16 bits will automatically be loaded into register 39.



**Power:** The Synaptron Micro has a 3.3V regulator on board. The voltage to power this regulator must be provided at the VIN pin (17). If you are operating a motor above 9-10VDC you should provide a regulated 5V value at VIN. External capacitance may also be added to provide a current source to the motor.



## Test Software:



After wiring up your system and verifying that you have communication with the Synaptron you can open "AN2011\_programming\_script.txt" using the Open File button in section #1 (highlighted above). Then press the Program button. This will program the new register settings for this application note.

To modify PID related settings during testing you can use the text boxes shown at the top of section #2. You can also modify the control and feedback associated settings in section #2.

To command position changes you can either write values to register 39 (*ControlInputLow*) in the serial test section (#1), or change the value in the Serial Ctrl text box (located in the Control Signal Settings frame) in section 2, and press the "Write" button.

## Register Settings-

Most register values were left at their default values. Those changed for this application note are noted below.

Index	Register Name	Default Value	Value for this app note
0	<i>FlashCycles</i>	9998	
1	<i>UnitAddress</i>	54	
2	<i>Command</i>	0	
3	<i>Function</i>	Changed	34
4	<i>Status</i>	0	
5	<i>PositionLow</i>	0	
6	<i>PositionHigh</i>	0	
7	<i>Velocity</i>	0	
8	<i>Acceleration</i>	0	
9	<i>NegativeLimitLow</i>	Changed	-30000
10	<i>NegativeLimitHigh</i>	0	
11	<i>PositiveLimitLow</i>	Changed	30000
12	<i>PositiveLimitHigh</i>	0	
13	<i>ControlLoopRate</i>	4	
14	<i>NegativePWMLimit</i>	-3685	
15	<i>PositivePWMLimit</i>	3685	
16	<i>PWMFrequency</i>	20000	
17	<i>PWMOutput</i>	0	
18	<i>MaxDutyCycle</i>	3685	
19	<i>PIDDivider</i>	Changed	50
20	<i>PTerm</i>	Changed	400
21	<i>ITerm</i>	Changed	2
22	<i>DTerm</i>	Changed	250
23	<i>ErrorBand</i>	0	
24	<i>AnalogSampleCount</i>	16	
25	<i>AnalogControl</i>	0	
26	<i>AnalogFeedback</i>	0	
27	<i>ControlSource</i>	0	
28	<i>ControlMultiplier</i>	1	
29	<i>ConctrolDivider</i>	1	
30	<i>ControlOffset</i>	0	
31	<i>ControlResultLow</i>	0	
32	<i>ControlResultHigh</i>	0	
33	<i>FeedbackSource</i>	Changed	3
34	<i>FeedbackMultiplier</i>	1	
35	<i>FeedbackDivider</i>	1	
36	<i>FeedbackOffset</i>	0	
37	<i>FeedbackResultLow</i>	0	
38	<i>FeedbackResultHigh</i>	0	
39	<i>ControlInputLow</i>	0	
40	<i>ControlInputHigh</i>	0	
41	<i>BaudValue</i>	3	
42	<i>Signal</i>	0	
43	<i>SignalTimeBase</i>	1	
44	<i>Ticks</i>	0	
45	<i>Current</i>	0	
46	<i>CurrentMultiplier</i>	129	
47	<i>CurrentDivider</i>	100	
48	<i>NegativeCurrentLimit</i>	-1500	
49	<i>PositiveCurrentLimit</i>	1500	
50	<i>IndexLow</i>	0	
51	<i>IndexHigh</i>	0	
52	<i>Function2</i>	0	
53	<i>VelocityLimit</i>	0	
54	<i>Reg54</i>	0	

55	Reg55	0	
----	-------	---	--

## **Register Changes Described:**

### **Function (register #3)-**

The *Function.F\_SoftLimits* bit (bit #1) is set which enables software limit points defined by the *NegativeLimitLow* (reg. #9), and *PositiveLimitLow* (reg. #11). Any commanded positions above the positive limit, or below the negative limit will be restricted.

Also set is the *Function.F\_EnableOnBrake* (bit #5), which causes the Synaptron to reset the H-bridge if a fault condition occurs. If this bit is not set then the operating system should periodically look for faults by checking bit #4 of the *Status* register.

### **Software Limit Registers (#9 and #11) -**

*NegativeLimitLow* (reg. #9), and *PositiveLimitLow* (reg. #11) are modified to create boundaries at -30000 and +30000 position counts.

### **PID Registers (#19-22)-**

The *PIDDivider*, *PTerm*, *ITerm*, and *DTerm* registers are modified to allow a PID algorithm to act on the error signal in your system. The error signal is the Commanded Position – Actual Position. In all likelihood you will want to modify these values depending on how they work with your mechanical system. More information on how the PID works can be found in the Synaptron user datasheet at [www.solutions-cubed.com](http://www.solutions-cubed.com).

### **Feedback Source (#33)-**

The *ControlSource* register (#27) defaults to serial control so it can remain unchanged for this application note. The *FeedbackSource* register defaults to open loop control (no feedback), and it must be changed to take its value from the quadrature encoder. Note that the position value determined by the encoder's pulses is stored in *PositionLow* (#5).

### **Additional Register Notes -**

To limit the speed of the motor as it moves from one point to another you can reduce the *NegativePWMLimit* and *PositivePWMLimit* register values.

Unused registers and inputs can be used for other application specific needs. For example, since the hardware limit switches are not being used they can act as button inputs read by a master unit to execute moves or other functions. Similarly the analog inputs could be used to measure switches or sensors.

The test software can be used to graph movements. Several movements were commanded via the serial interface (writing to register 39) with the settings described in this application note.

