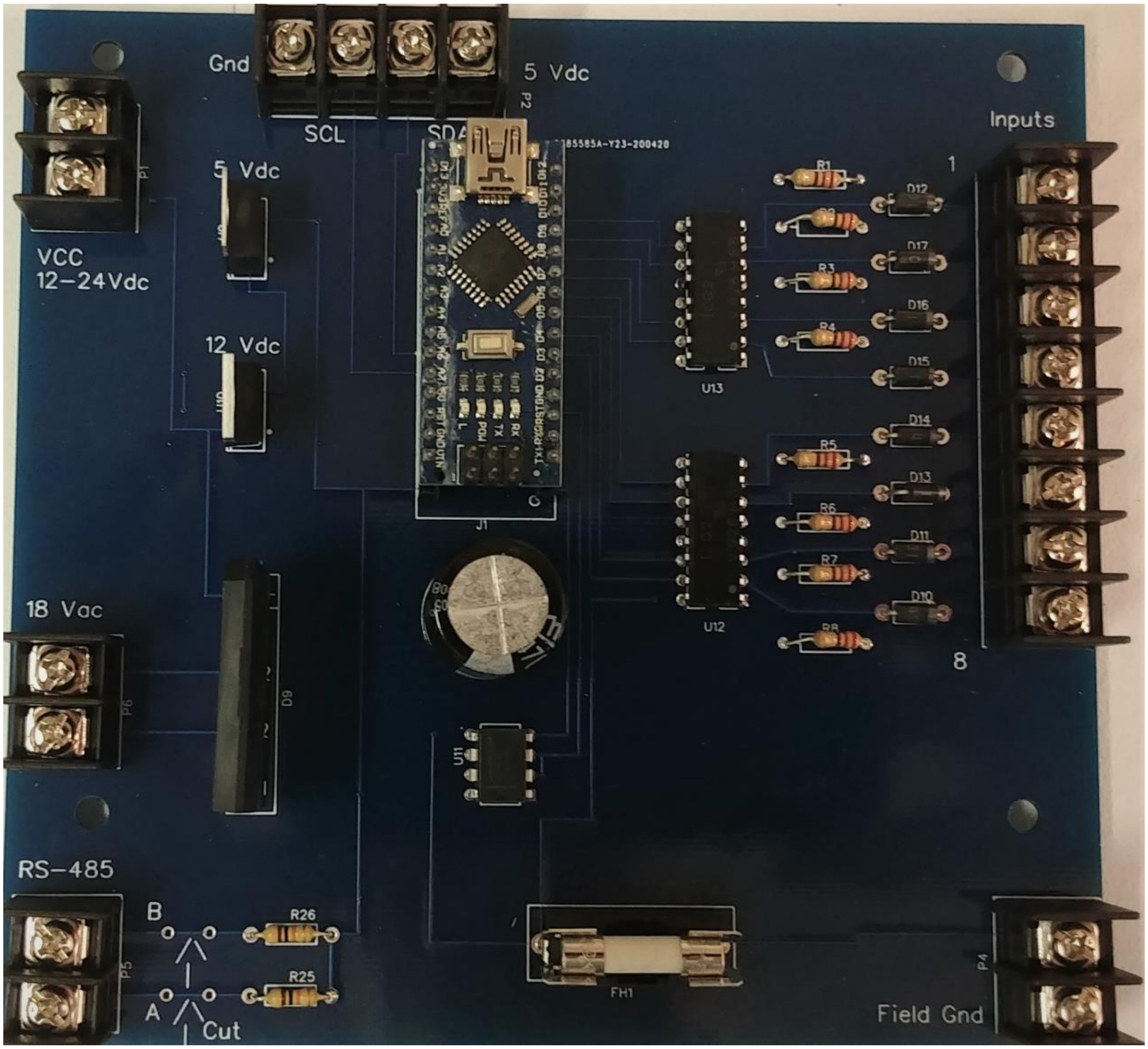
C/MRI 8 Inputs w/ I2C Card

This card is designed as a generic 8 inputs and an I2C communication bus card. The inputs are active low (you apply ground) and the I2C bus logic is written to control servos used for turnouts.



Along the left edge the top terminal strip is for 12 to 24 VDC and the middle terminal strip is for 18 VAC. These are the two types of supply power the board can use. **CAUTION: You must choose just one of these to supply power, never connect both types of power at the same time.**

The bottom terminal strip (still left edge) is the RS-485 connection. The Arduino is programmed to use this port as a C/MRI node (Computer / Model Railroad Interface) and will connect directly to JMRI (Java Model Railroad Interface) as such.

Just to the right of the RS-485 connection is a silk screen label that says “cut here.” This feature is used in conjunction with how many cards you are using. If you are using a single card on the RS-485 connection leave the card as delivered, if however you use more than one card you will need to cut this PCB trace on all but the first card. More details to follow.

To the bottom right (shown horizontally) is the 1.6 Amp fuse. This fuse protects the field devices from drawing excessive current and the card from short circuits.

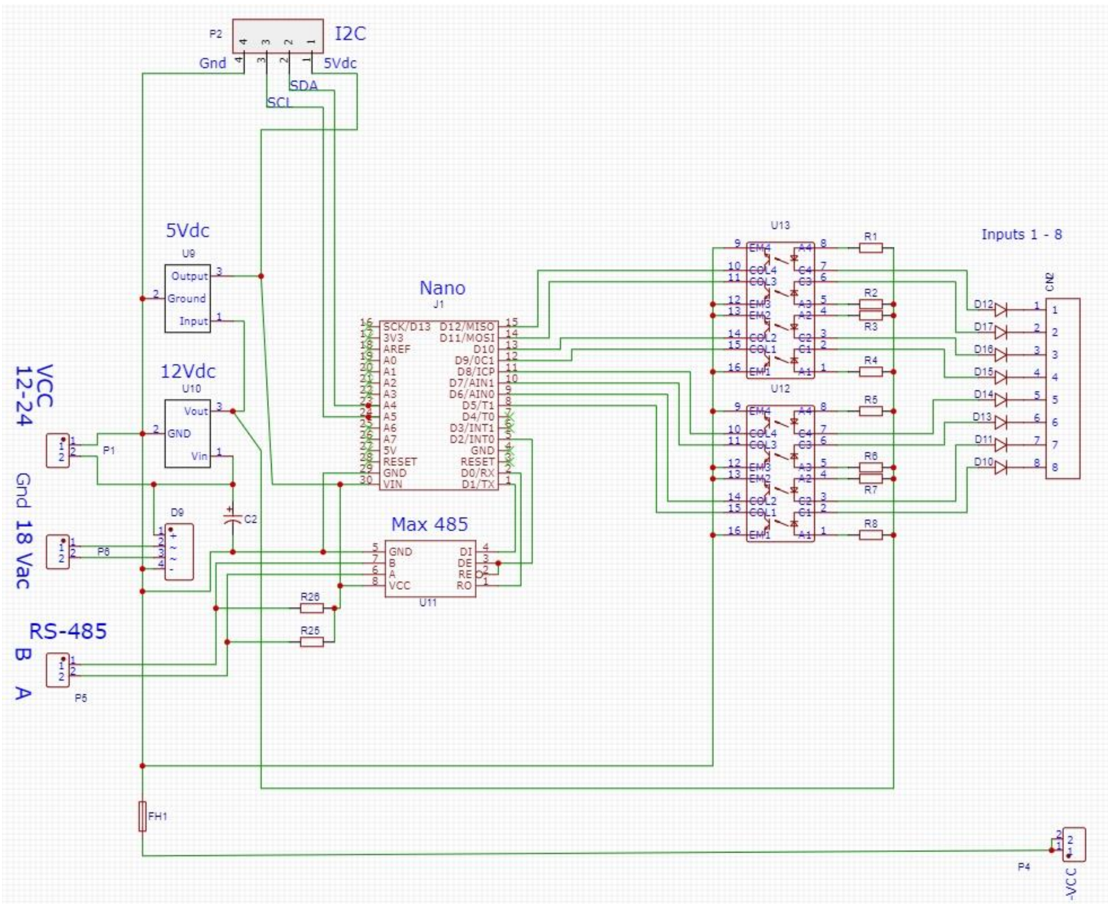
The bottom right edge terminal strip is the Field Ground connection. This terminal is provided as a convenient point to source your field devices. More details to follow.

The top right edge terminal strip is the eight input terminals. Route the field ground through your device and back to one of these inputs.

The Top edge (left side) is the I2C (pronounced eye-squared-see) terminal strip. NOTE: the Gnd & 5 VDC is used to power the PWM board logic circuit and not its power output circuits. More details to follow.

Schematic Overview:

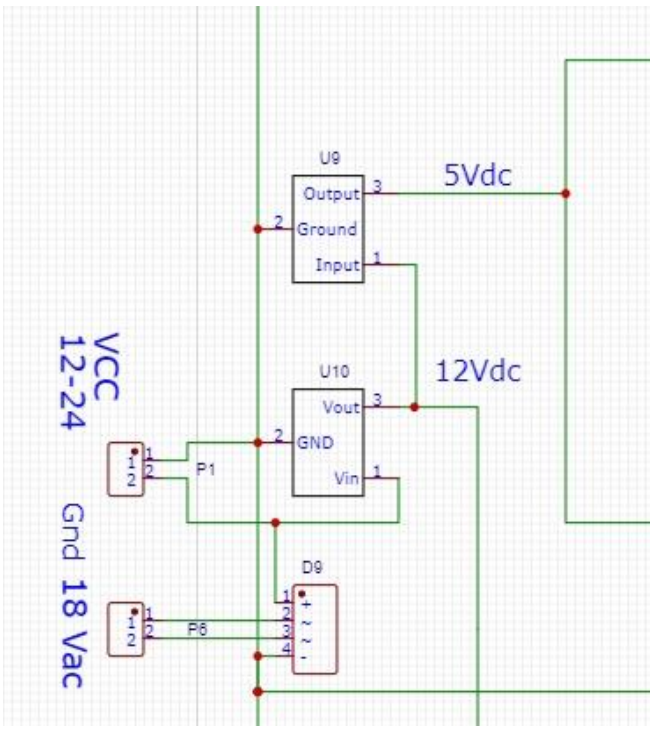
Please don't be alarmed by this schematic, mastery is not necessary to use the C/MRI siding card. It is included for a more in-depth understanding of how the card works and will allow advanced users to adapt the card for other purposes.



Blow up the PDF page for more clarity

Power sources:

The board may be run from either a 12-24 VDC or an 18 VAC power source (not both.)



18 VAC – Is applied to terminal strip P6 and connected to a full bridge rectifier D9 (terminals 2 & 3) where it is converted to approximately 25 VDC. The output of the rectifier is applied to a regulator (U10 pin 1) which limits the output (pin 3) to 12 VDC.

The 12 VDC is used in two places:

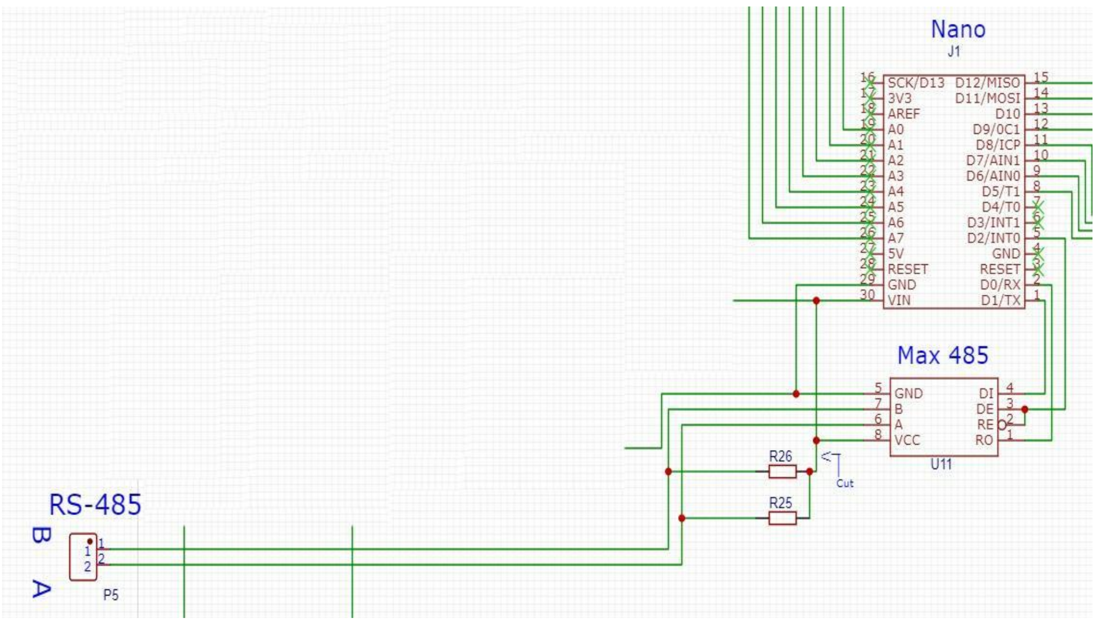
1. Power to source the outputs
2. The input to U9 (pin 1), the 5 VDC regulator where the output (pin 3) is used for the internal board logic.
3. NOTE: D9-4, U10-2 and U9-2 form a common ground.

12-24 VCC – Is applied to terminal strip P1 and connected to the 12 VDC regulator (U10) which in-turn feeds the 5VDC regulator (U9)

CAUTION: Never connect both 18VAC and 12-24VDC to the same board at-the-same-time as this will damage the rectifier (D9) and 12VDC regulator (U10) and possibly the externally connected power sources.

RS-485 Connection:

This is the connection that allows the board to communicate with and external control device, such as a computer running JMRI (Java Model Railroad Interface.) The Arduino on this card uses libraries to make it communicate as a C/MRI card (Computer / Model Railroad Interface.)

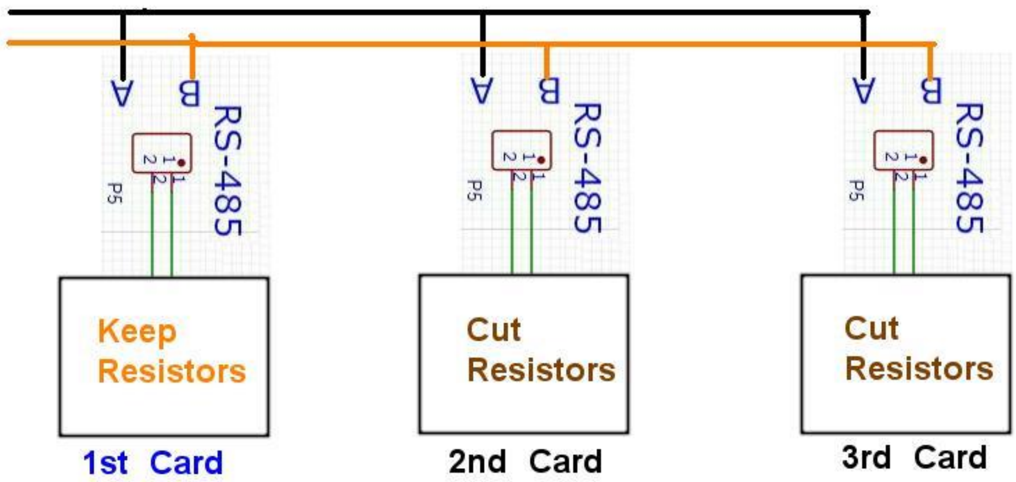


The computer connects at the terminal strip labeled RS-485. The computer will probably have a USB-to-RS-485 adapter. Make sure to follow the “A” and “B” wiring so that “A” goes to “A” and “B” goes to “B”.

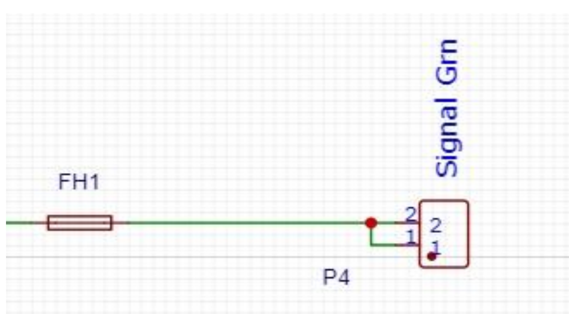
The communications bus needs what are called pull-up resistors to ensure that no noise (unwanted + to - transitions) are seen by the Max 485 chip. Here they are identified as R26 and R25 but the numbers may vary depending on which card you are using. The important thing to know is that the “bus” only needs one set of resistors. So we want to keep the resistors on the first board and disable them on all additional boards. Each card will have a silk trace “Cut” or “Cut Here” point. (Depending on the card this could be labeled on the top-side or the bottom-side of the card.)

If you only have one card on the bus you are done – do nothing with the resistors. If you have two or more cards on the bus then you must,

1. Leave the resistors on the first board untouched (do not cut)
2. On each additional card you must disable these resistors by cutting the copper trace at the point labeled “Cut” or “Cut Here.” An X-Acto knife works well for this and the gap only needs to be wide enough to see that the trace has been cut.



Signal Ground:

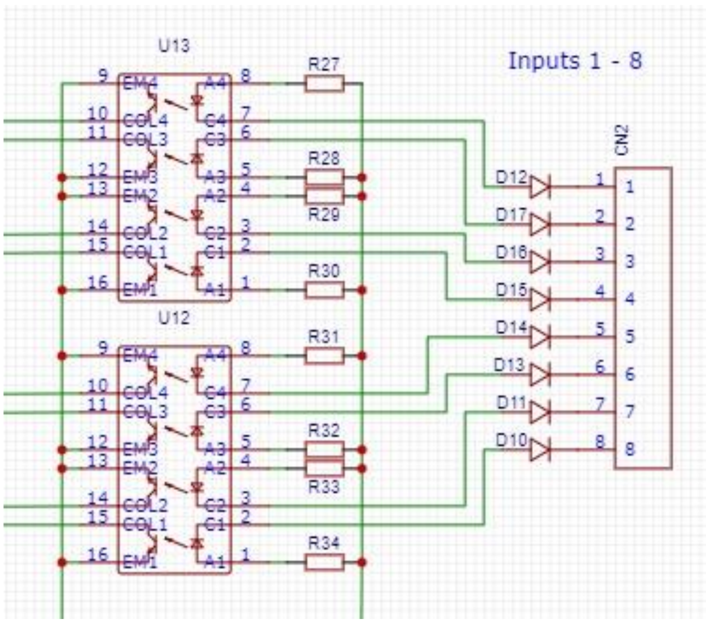


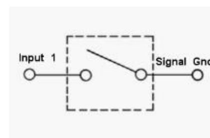
The Signal GND terminal strip (P4) terminals (1) and (2) are jumpered together on the board. Use these terminals to provide the connection to the common side of the field devices to be controlled. You can also use this ground to connect the common side of the input devices.

The board comes with a 1.6 Amp fuse but it is safe to use a more common 2 Amp fuse. If this fuse blows then most likely a field device has been shorted to 12VDC. Nothing on the card will blow this fuse.

Inputs 1-8:

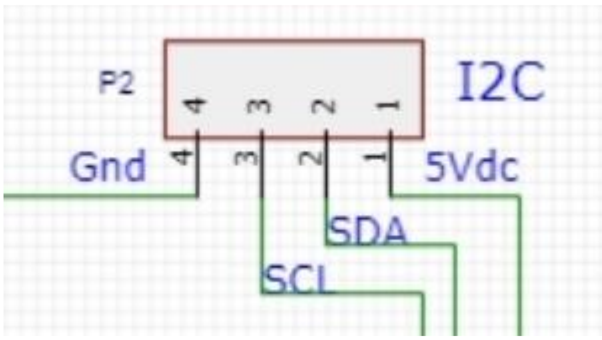
There are eight inputs on the board on terminal block CN2. Input 1 is on terminal (1) and Input 8 is on terminal 8. Examining Input 1 we see diode D12, this is a reverse polarity protection diode. D12 is connected to U13 pin (7) this chip provides galvanic isolation for the Nano input terminals. Resistor R27 on pin (8) is a current limiting resistor. Input 1 is active when CN2 pin (1) is connected to Signal GND (active low.)





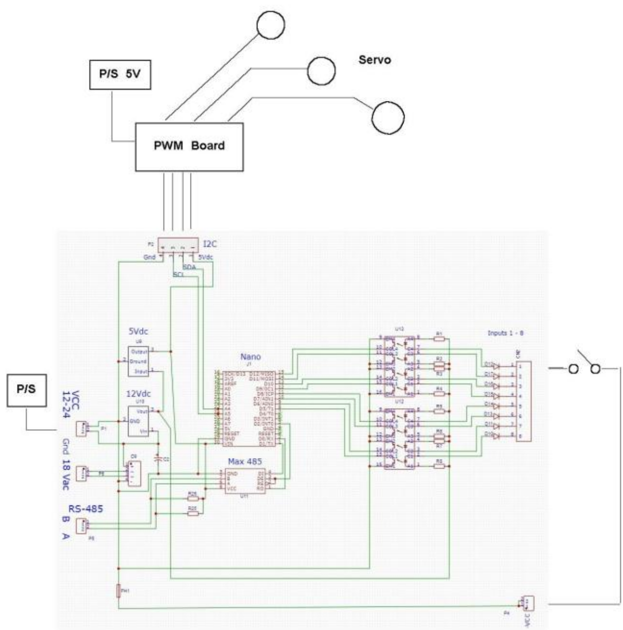
I2C Outputs:

Details on how this communication bus works is beyond the scope of this manual but for more advanced users the bus will support any I2C device. The logic in the Arduino is written to support servos used to control turnouts. The result is a servo changes a turnout from “Closed” to “Thrown” and vice-versa. Advanced users could rewrite the Arduino code to support other devices.



Gnd (ground) and the 5 Vdc is supplied to run the connected device logic only. The onboard 5VDC regulator cannot supply enough power to run most auxiliary devices. SCI provides the clock signal while SDA sends the data to the next device,

This card expects to see a PCA9685 16-Channel 12-bit PWM Servo Motor Driver I2C Module. The Arduino Logic is written to support this card. Details will follow.



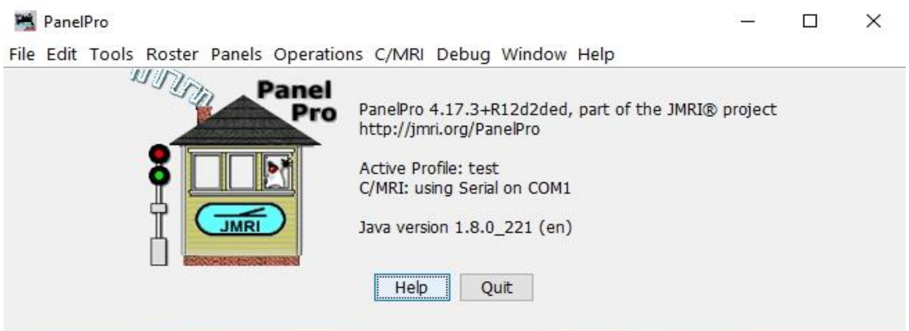
This is an example of a typical input and output connection.

Note how the board “Field Ground” connection is on the common side for all connections.

Configuring JMRI to use the board as C/MRI hardware:

The on-board Arduino chip is programmed to act as a C/MRI SMINI card with the default address of zero (0). Though a true SMINI card can address 24 inputs and 48 outputs the Nano chip cannot handle that much I/O. This input / output board can only handle a total of 16 I/O points, so all SMINI node I/O points beyond sixteen (16) will be ignored by the Nano chip.

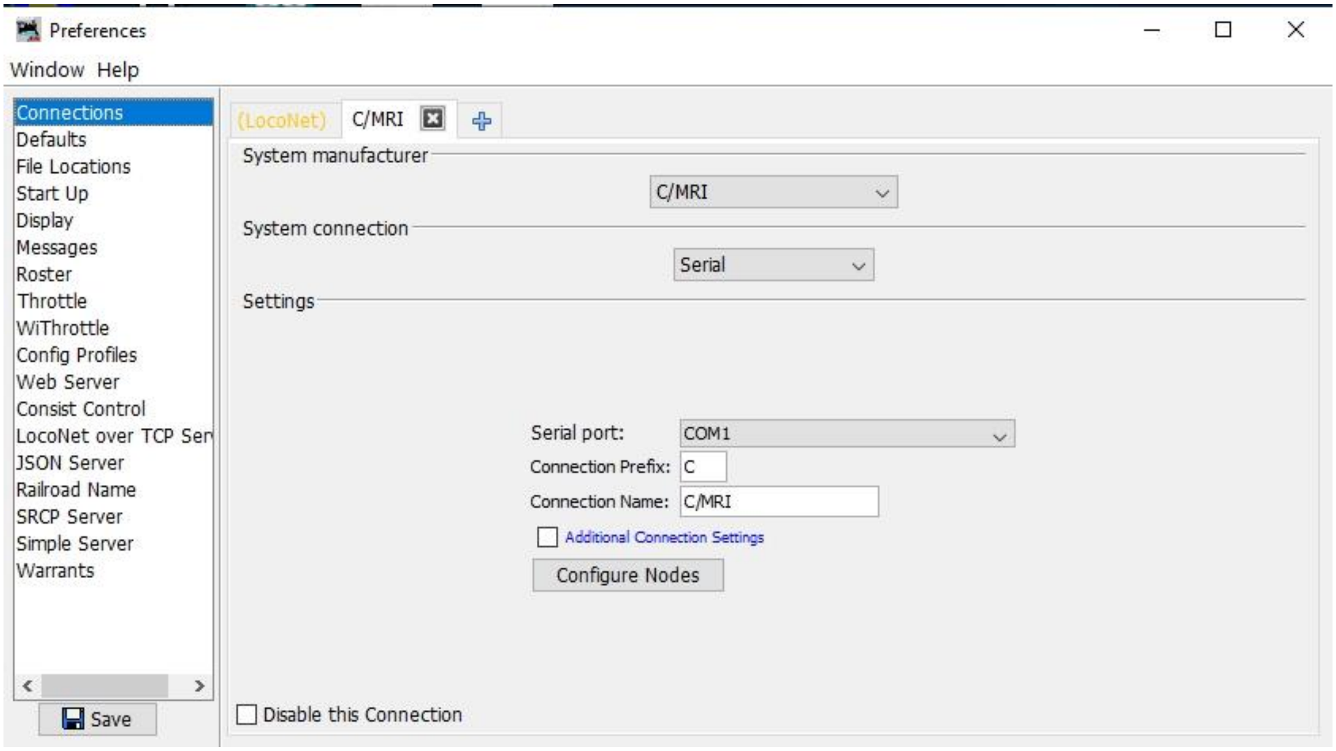
This overview of how to program a C/MRI port is not a comprehensive guide to JMRI but it should be enough information to get you started.



Note that this manual is written using PanelPro version 4.17.3. Some screens may vary depending on your version of PanelPro.

When starting this example your splash screen will not show the C/MRI on COM1. However as we proceed you will be instructed that PanelPro must be restarted and that is when you'll see the C/MRI listing. NOTE: the menu at the top of this splash screen as all references to menu selections will start from this window.

Go to **Edit/Preferences** on this menu to get the following pop-up.



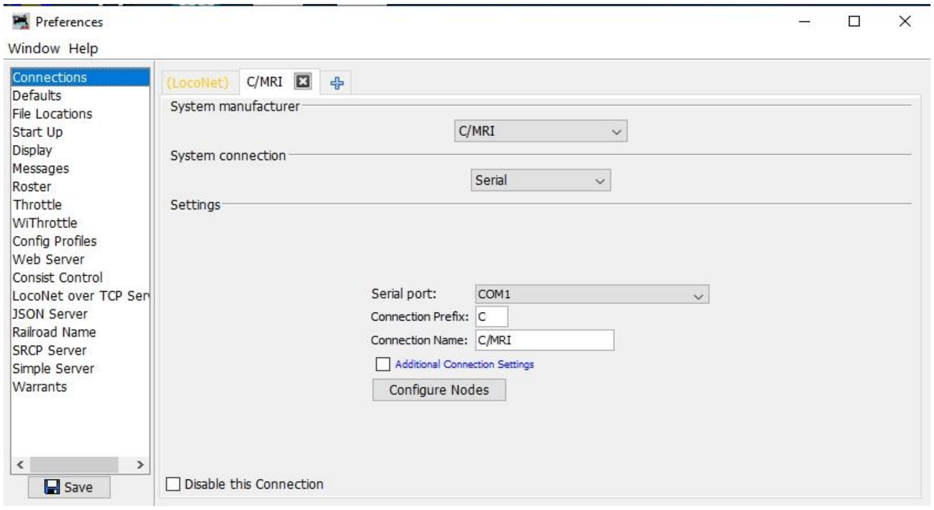
Now select connections from the side-bar menu.
Your window will now have at least two tabs, one is usually named for the DCC system you are using and the other has a big "plus" sign. Click the "plus" sign to add a new connection for JMRI to use.
From the drop-down list boxes select the following:

- System Manufacturer – C/MRI
- System Connection – Serial
- Serial Port – COM1 (this may vary depending on how many COM ports your computer has used.

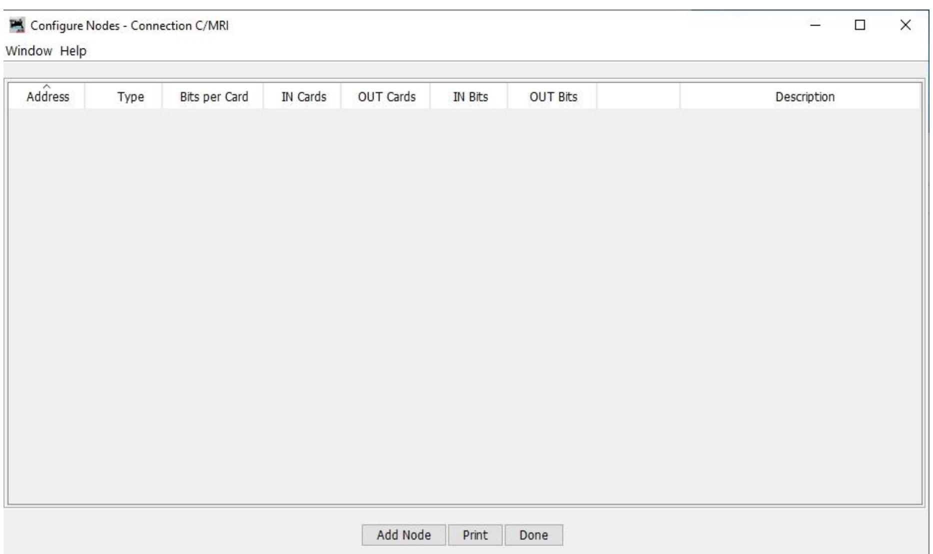
Leave the connection prefix with the default "C."
The Connection Name can be anything you want it to be, I suggest C/MRI as this is the name that will show up on the tab and in other menu choices making it easy to remember what hardware you are talking too.
Additional Connection Settings: Set the baud rate to 9600

Once you fill in the blanks you must click the SAVE button. A pop-up we tell you that PanelPro must be restarted for the changes to take effect. Answer YES and now you'll see the splash screen as shown above.

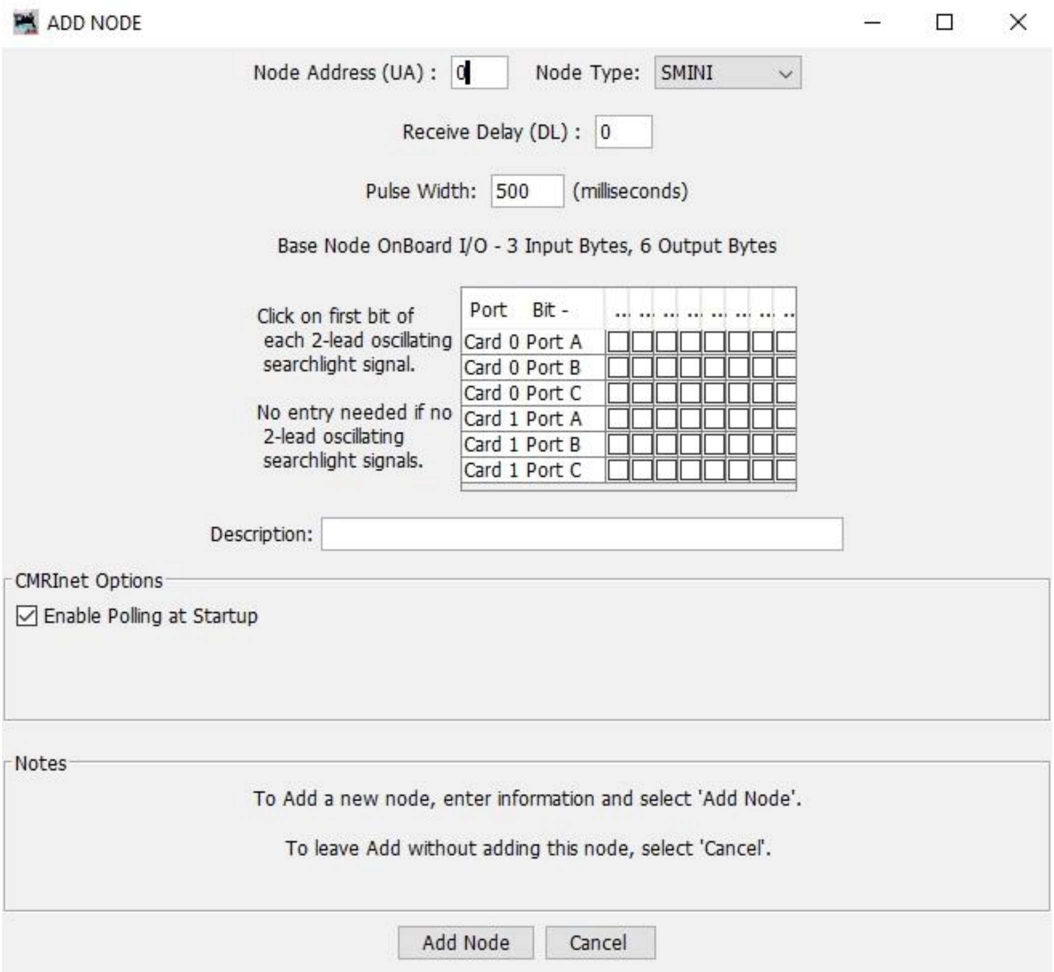
From the splash screen menu go to EDITS/PREFERENCES



Once again select Connections from the side-bar menu. Click the “Configure Nodes” button



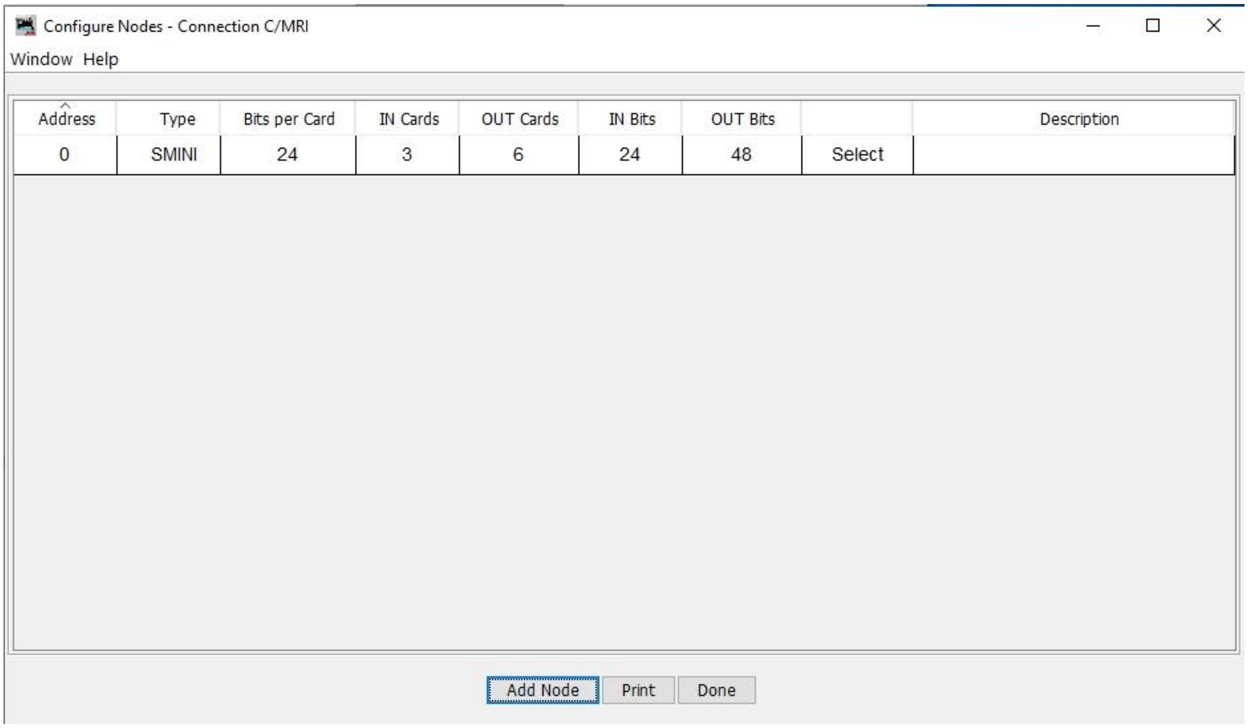
A new blank node list pop-up will be displayed. Click the “Add Node” button to open the next pop-up



C/MRI nodes start at address zero (0) so set that address (the next node if needed would be one (1)) and set the Node Type to SMINI and leave the rest of the boxes set with their default values.

NOTE: You may wish to add a Description to help you remember what this node is used for.

Click to "Add Node" button to close this pop-up

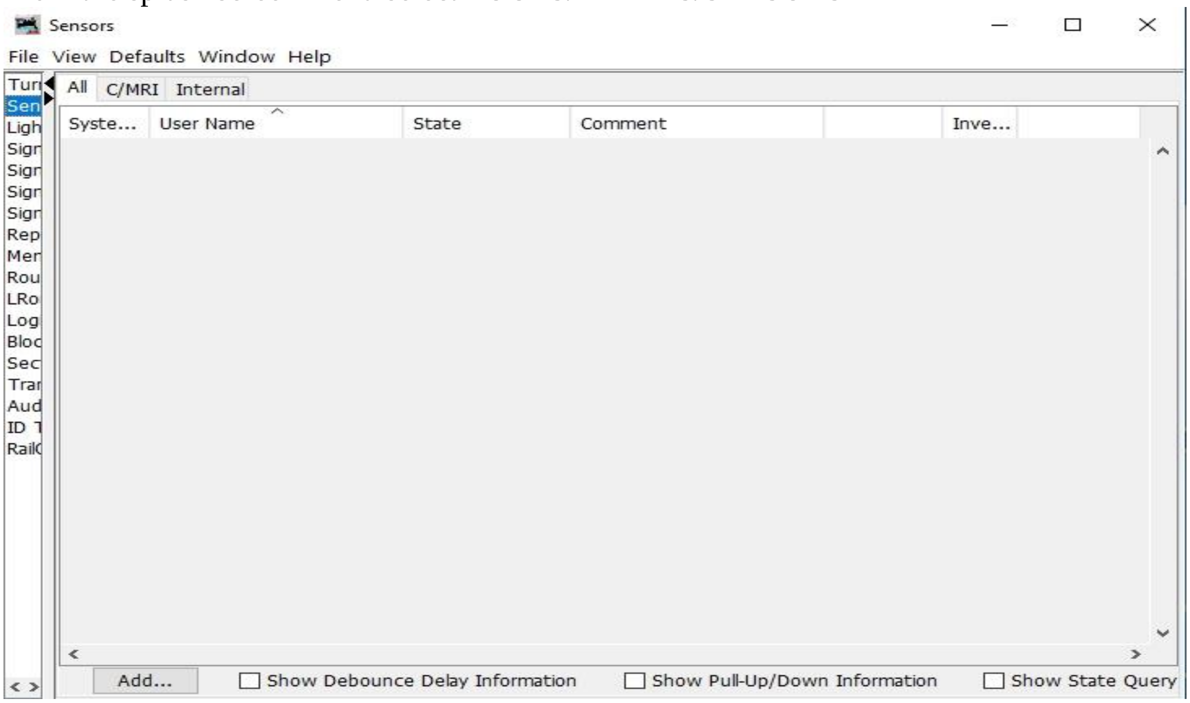


The node configuration pop-up now shows the node you just added. Double check the address (0) and type SMINI before clicking the “Done” button to close this pop-up.

We now have a C/MRI node configured and ready for JMRI to use.

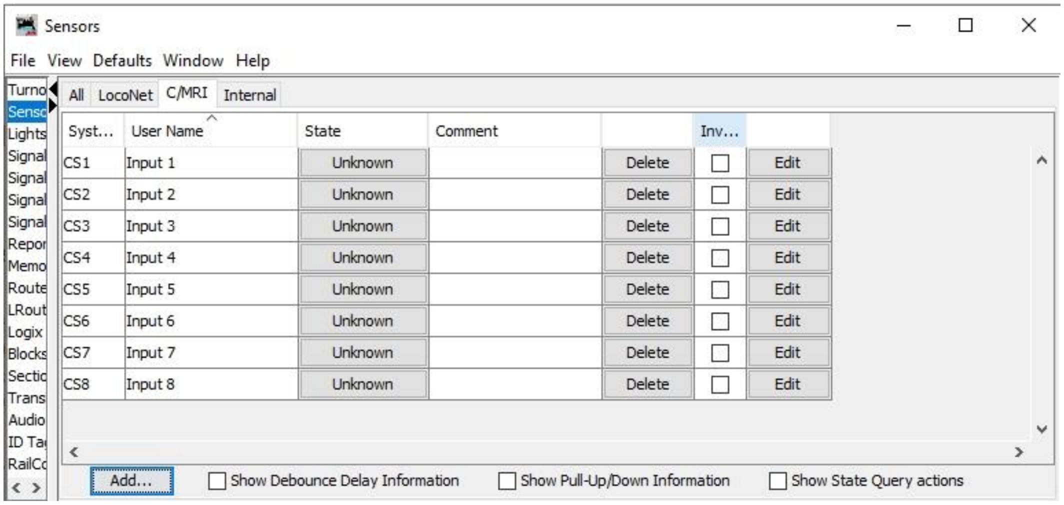
Adding the Arduino inputs to JMRI sensors

From the splash screen menu select TOOLS/TABLES/SENSORS



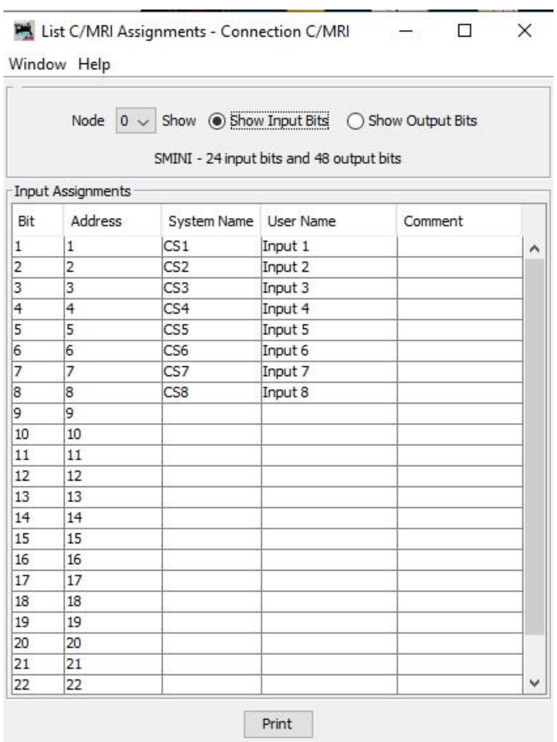
Sensors are input signals for JMRI to use for status displays and logic decisions. We have eight (8) inputs on the card to define. Click the "Add" button to get started.

Make sure that the System Connection is set to C/MRI. C/MRI addresses are 4 digits long. The first zero (0) (and sometimes the second) indicates the node being addressed. Bit addresses start with one (1) and again we'll use only node zero (0) so our addresses will be 0001 – 0008.
We start with Hardware Address 0001 and give it a User Name of our choosing ("Input 1" is my demo name.) Press the "Create" button and the pop-up will automatically close. Repeat this process to add the other seven inputs as shown below.



Syst...	User Name	State	Comment		Inv...	
CS1	Input 1	Unknown		Delete	☐	Edit
CS2	Input 2	Unknown		Delete	☐	Edit
CS3	Input 3	Unknown		Delete	☐	Edit
CS4	Input 4	Unknown		Delete	☐	Edit
CS5	Input 5	Unknown		Delete	☐	Edit
CS6	Input 6	Unknown		Delete	☐	Edit
CS7	Input 7	Unknown		Delete	☐	Edit
CS8	Input 8	Unknown		Delete	☐	Edit

From the Splash Screen got to C/MRI/List Assignments



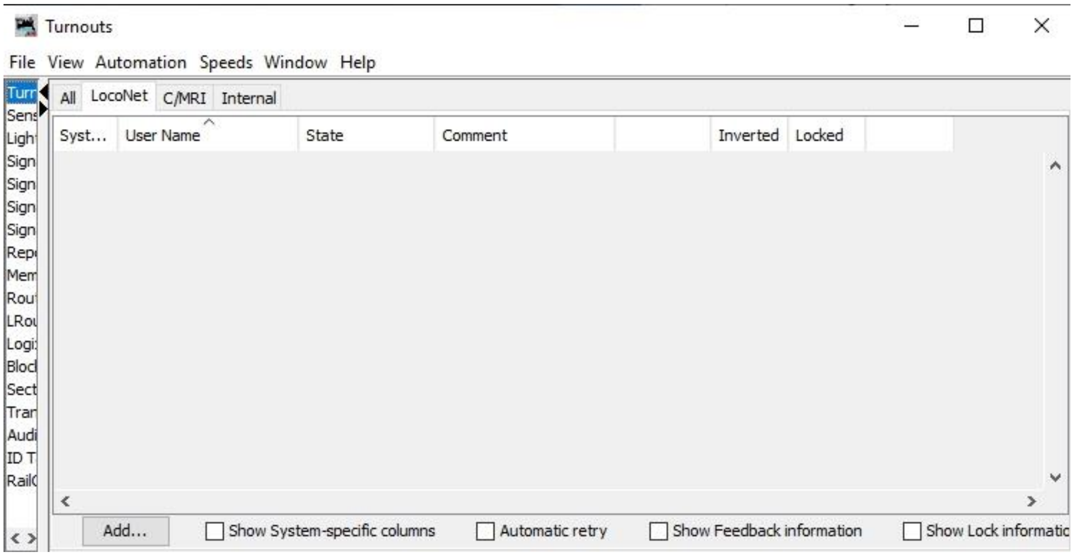
Bit	Address	System Name	User Name	Comment
1	1	CS1	Input 1	
2	2	CS2	Input 2	
3	3	CS3	Input 3	
4	4	CS4	Input 4	
5	5	CS5	Input 5	
6	6	CS6	Input 6	
7	7	CS7	Input 7	
8	8	CS8	Input 8	
9	9			
10	10			
11	11			
12	12			
13	13			
14	14			
15	15			
16	16			
17	17			
18	18			
19	19			
20	20			
21	21			
22	22			

Check that you're on "Node 0"
Click "Show Input Bits"

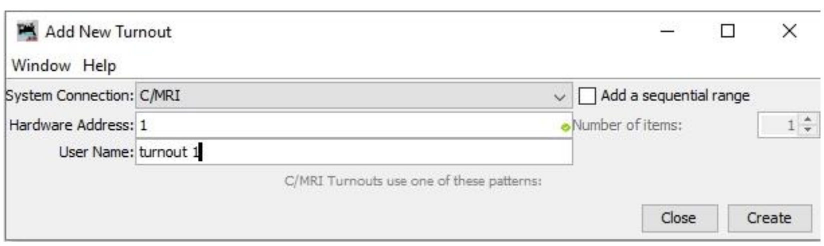
Here we can see that JMRI has linked its internal "System Name" and "User Name" to bits in the C/MRI hardware.

Adding the Arduino I2C to JMRI turnouts

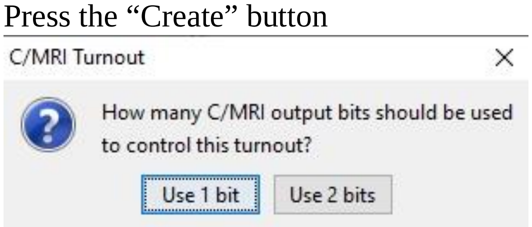
From the Splash Screen got to Tools/Tables/Turnouts



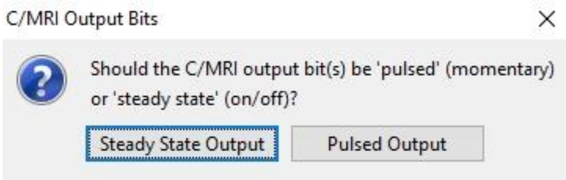
Turnouts are output signals from JMRI used to control the state of each turnout. We have sixteen (16) turnouts on the card to define. Click the "Add" button to get started.



Make sure that the System Connection is set to C/MRI. C/MRI addresses are 4 digits long. The first zero (0) (and sometimes the second) indicates the node being addressed. Bit addresses start with one (1) and again we'll use only node zero (0) so our addresses will be 0001 – 0016.
We start with Hardware Address 0001 and give it a User Name of our choosing ("turnout 1" is my demo name.)
Press the "Create" button

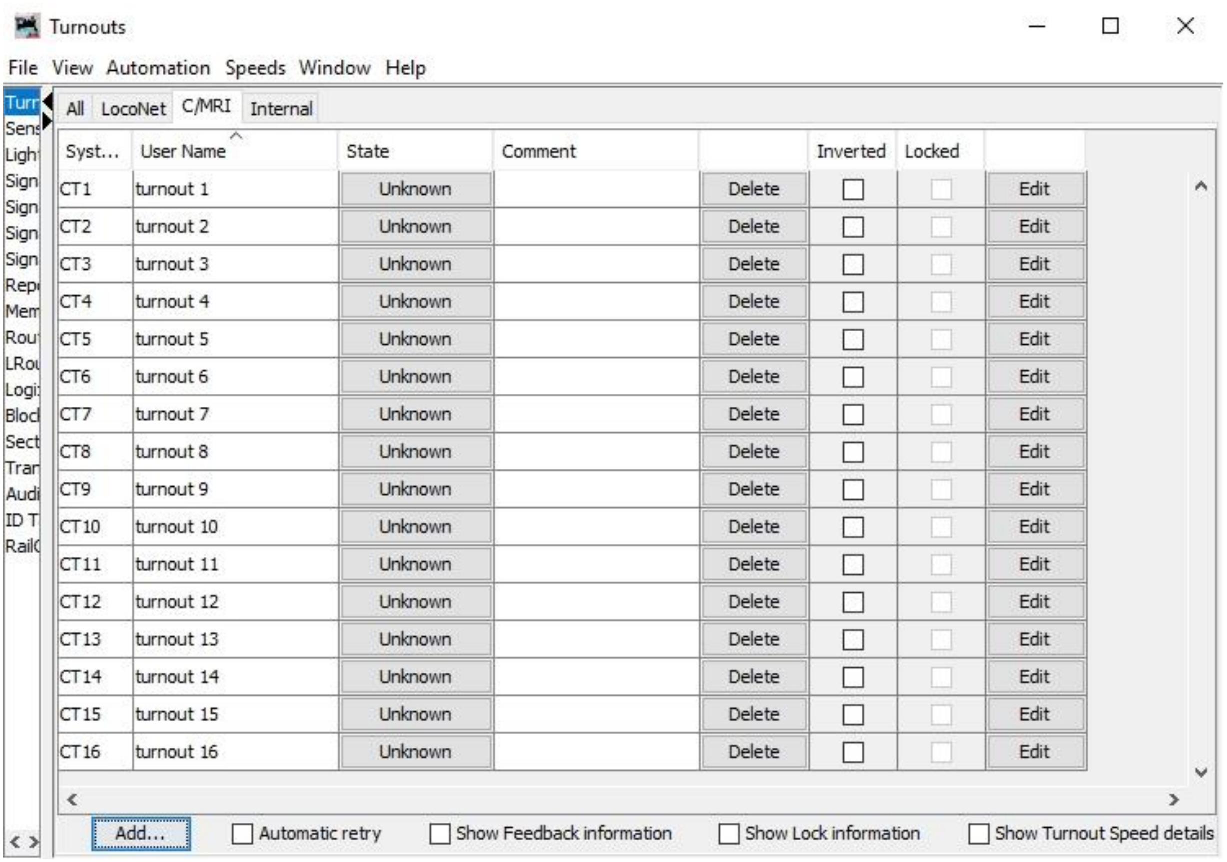


For theses outputs we'll use just a single bit for control. Bit on = thrown while bit off = closed. Click - Use 1 bit.

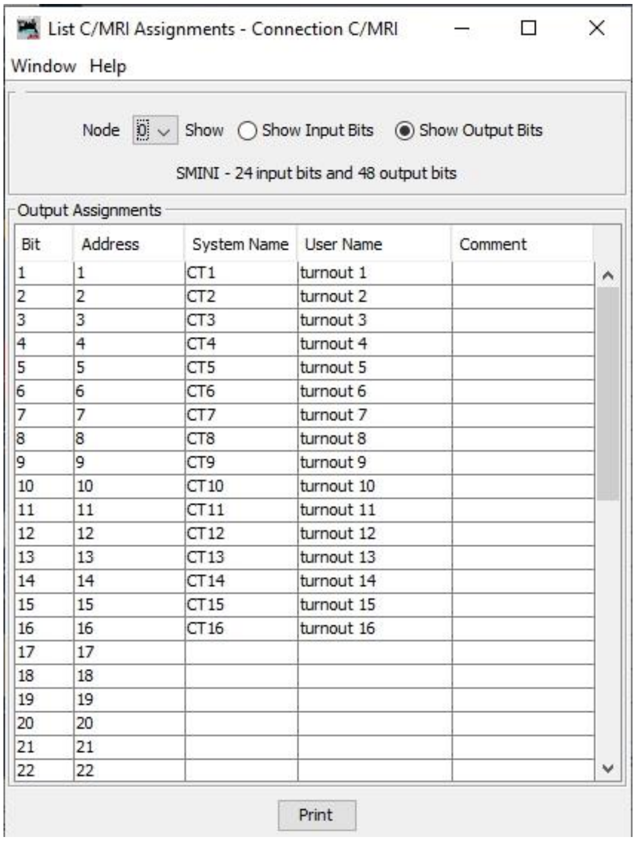


The next pop-up wants to know how we want to use this bit. We want the bit to be on constantly when select and off constantly when not selected. Click - Steady State Output. The pop-ups will now close adding the new turnout to the table.

Repeat this process to add the other fifteen turnouts as shown below.



From the Splash Screen got to C/MRI/List Assignments



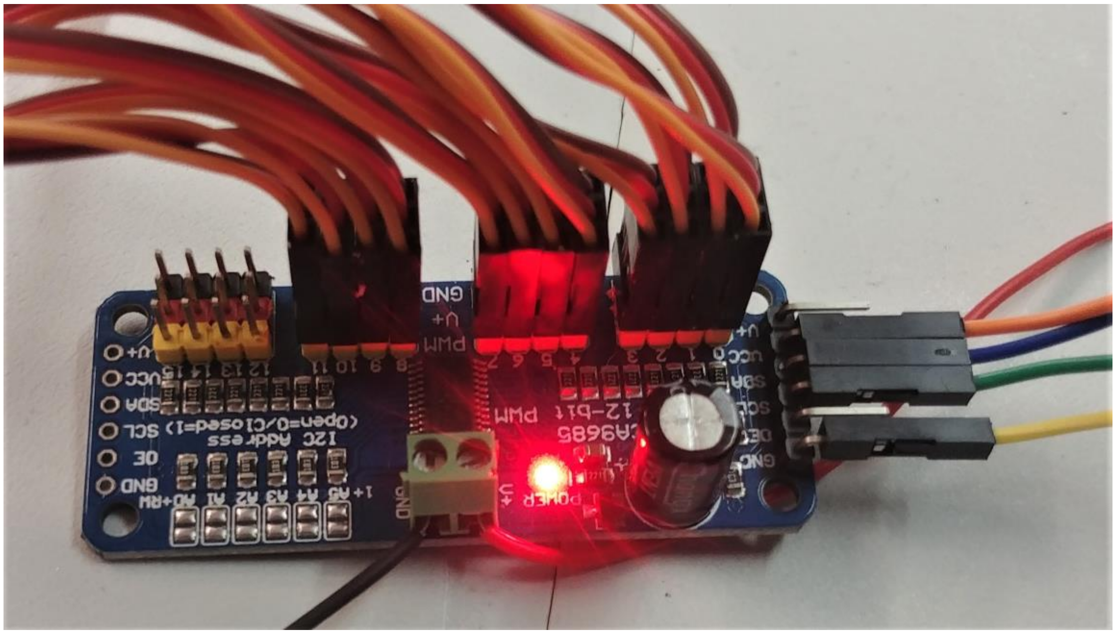
Bit	Address	System Name	User Name	Comment
1	1	CT1	turnout 1	
2	2	CT2	turnout 2	
3	3	CT3	turnout 3	
4	4	CT4	turnout 4	
5	5	CT5	turnout 5	
6	6	CT6	turnout 6	
7	7	CT7	turnout 7	
8	8	CT8	turnout 8	
9	9	CT9	turnout 9	
10	10	CT10	turnout 10	
11	11	CT11	turnout 11	
12	12	CT12	turnout 12	
13	13	CT13	turnout 13	
14	14	CT14	turnout 14	
15	15	CT15	turnout 15	
16	16	CT16	turnout 16	
17	17			
18	18			
19	19			
20	20			
21	21			
22	22			

Check that you're on "Node 0"
Click "Show Output Bits"

Here we can see that JMRI has linked its internal "System Name" and "User Name" to bits in the C/MRI hardware.

JMRI is now configured to work with C/MRI node 0 and will now poll COM port 1 to send and receive I/O commands and status.

The PCA9685 Adafruit PWM/Servo Driver



This board requires no special set-up to be used with the I2C board interface. On the right hand side are the 4 I2C connections (GND, SCL, SDA VCC) the V+ connection is not used. Connect these terminals to the matching terminals on the board connector P2 also labeled I2C

Connect an external 5VDC power source to the green connector. This power source will drive the servo motors. For a complete description of how this board works go to https://learn.adafruit.com/16-channel-pwm-servo-driver?view=all

The 8 input I2C card has been tested with the PCA9685 driver and the SG90 Micro Servo motor.

Arduino Code

```
#include <Auto485.h>
#include <CMRI.h>
//Author: Michael Adams (<http://www.michael.net.nz>)
// Copyright (C) 2012 Michael D K Adams.
// released under the MIT license.

#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
// create an object named pwm
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

#define SERVOMIN  125 // this is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX  575 // this is the 'maximum' pulse length count (out of 4096)

// test holds the value from CMRI output bits
boolean test = LOW;

// assign eight input names to Arduino pin numbers
#define input_1 12
#define input_2 11
#define input_3 10
#define input_4 9
#define input_5 8
#define input_6 7
#define input_7 6
#define input_8 5

//create a bus object using pin 2 for control
#define DE_PIN 2
Auto485 bus(DE_PIN);

// create a cmri object, give it an address + type of CMRI card + communicate on bus
#define CMRI_ADDR 0
CMRI cmri(CMRI_ADDR, 24, 48, bus);

void setup() {

    // start pwm and assign a frequence
    pwm.begin();
    pwm.setPWMFreq(60);  // Analog servos run at ~60 Hz updates
    yield();
```

```
    // start the bus communication
    bus.begin(9600,SERIAL_8N2);

    // tell Arduino how pins will be used
    pinMode(input_1, INPUT_PULLUP);
    pinMode(input_2, INPUT_PULLUP);
    pinMode(input_3, INPUT_PULLUP);
    pinMode(input_4, INPUT_PULLUP);
    pinMode(input_5, INPUT_PULLUP);
    pinMode(input_6, INPUT_PULLUP);
    pinMode(input_7, INPUT_PULLUP);
    pinMode(input_8, INPUT_PULLUP);

}

void loop() {

    // wake up cmri object
    cmri.process();

    boolean a = LOW;
    boolean b = LOW;

    // FOR cycles through 8 Arduino inputs
    // and saves the value in a
    for (int i = 0; i <= 7; i++) {
      switch (i) {
        case 0:
            a = digitalRead(input_1);
            break;
        case 1:
           a = digitalRead(input_2);
           break;
        case 2:
           a = digitalRead(input_3);
           break;
        case 3:
           a = digitalRead(input_4);
           break;
        case 4:
           a = digitalRead(input_5);
           break;
        case 5:
           a = digitalRead(input_6);
           break;
```

```
          case 6:
            a = digitalRead(input_7);
            break;
          case 7:
            a = digitalRead(input_8);
            break;
          default:
            // statements
            break;
        }
        // IF - ELSE inverts the Arduino bits
        // and stores the new value in b
        if (a == LOW) {
          b = HIGH;
        }
        else {
          b = LOW;
        }
        // i = cmri bit number b = value to send
        // set_bit passes value to cmri object so it
        // can be passed out to JMRI
        cmri.set_bit(i,b);
      }

    // FOR cycles through 16 output bits in the cmri object
    // sent to it from JMRI
    // and stores the value in test
    for (int i = 0; i <= 15; i++) {
        test = cmri.get_bit(i);
        // IF sets servo number (i)
        if (test==HIGH) {
            pwm.setPWM(i, 0, SERVOMAX );  // Closed
        } else {
            pwm.setPWM(i, 0, SERVOMIN );  // Thrown
        }
}
}
```