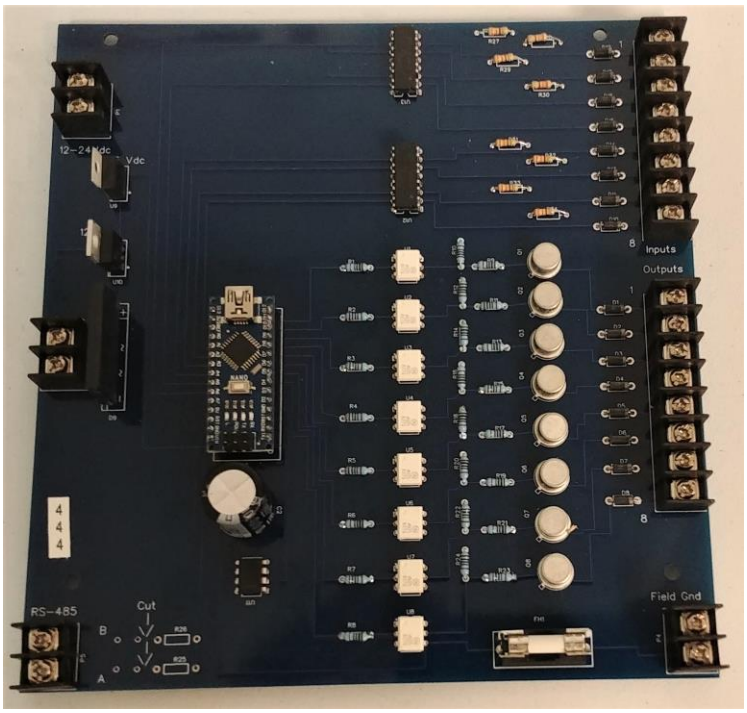
C/MRI 8 Inputs and 8 Outputs Card

This card is designed as a generic 8 inputs and 8 outputs interface card. The inputs are active low (you apply ground) and the outputs are Open Emitter (Terminal provides +12 VDC.) It is licensed and distributed under the MIT software license (https://opensource.org/licenses/MIT) and the TAPR OHL hardware license (https://tapr.org/the-tapr-open-hardware-license/ .)

Top View



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.

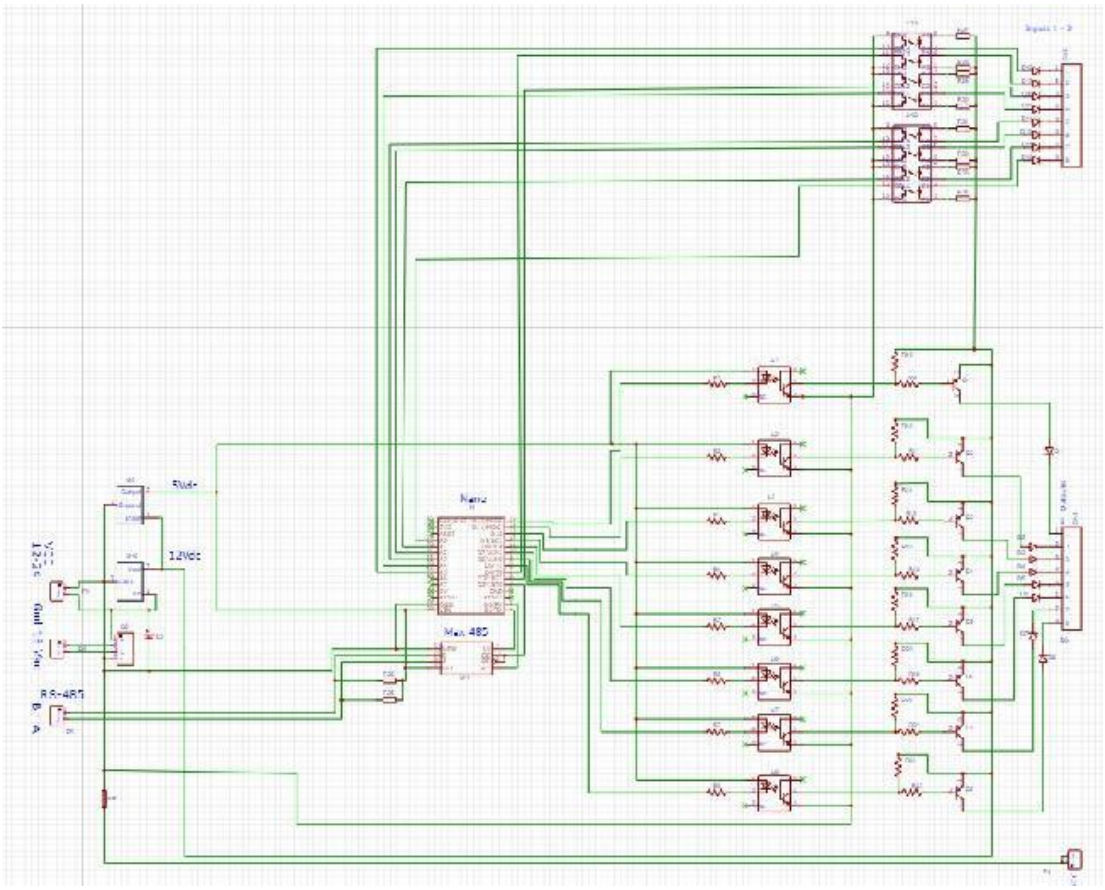
The bottom right edge terminal strip is the Field Ground connection. This terminal is provided as a convenient point to source your field devices. More details to follow.

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 middle right edge terminal strip is the eight output terminals. Route the field ground to the common connection of your device and route the control side (On/Off) side of the device back to the selected output terminal. The output will switch On/Off the +12 VDC power.

Schematic Overview:

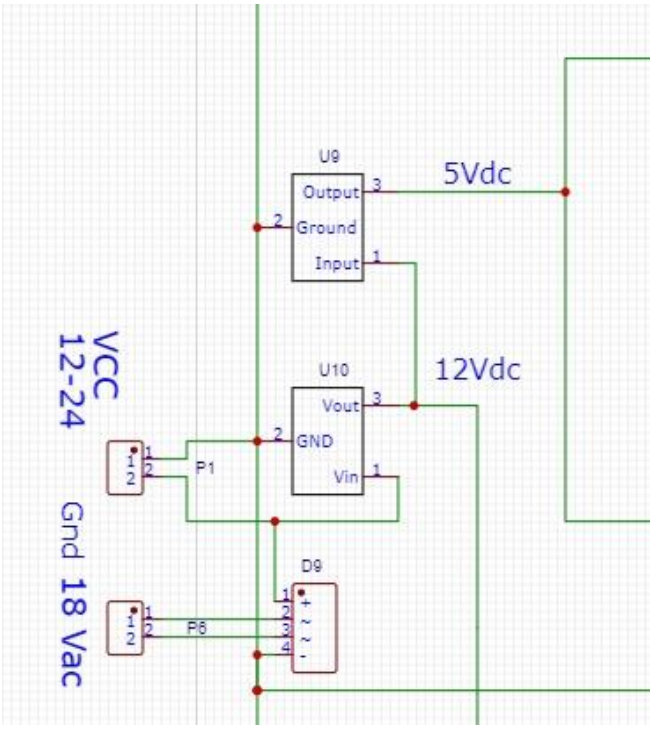
Please don't be alarmed by the following schematic, mastery is not necessary to use the C/MRI siding card. It is included for a more in-depth understanding of how the card works and will allow advanced users to adapt the card for other purposes.



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:

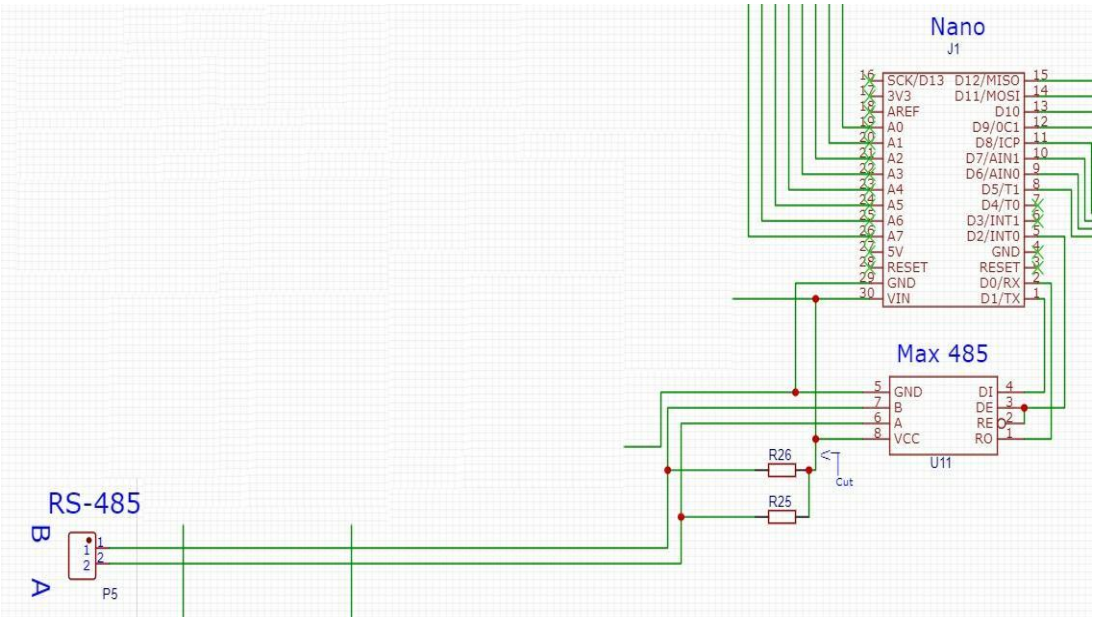
1. Power to source the outputs
2. The input to U9 (pin 1), the 5 VDC regulator where the output (pin 3) is used for the internal board logic.
3. NOTE: D9-4, U10-2 and U9-2 form a common ground.

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.)

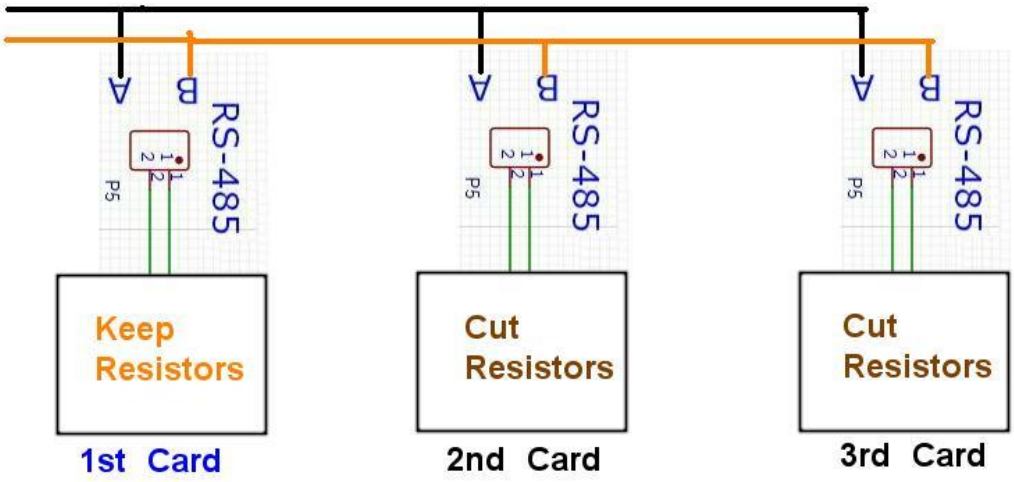


The computer connects at the terminal strip labeled RS-485. The computer will probably have a USB-to-RS-485 adapter. Make sure to follow the “A” and “B” wiring so that “A” goes to “A” and “B” goes to “B”.

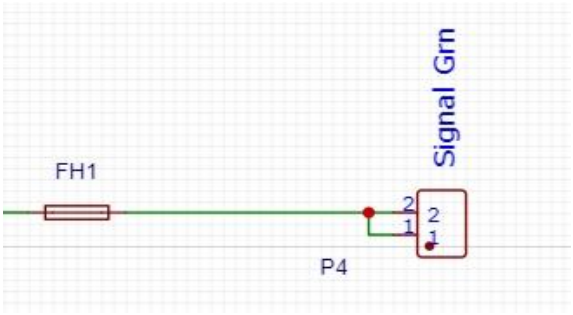
The communications bus needs what are called pull-up resistors to ensure that no noise (unwanted + to - transitions) are seen by the Max 485 chip. Here they are identified as R26 and R25 but the numbers may vary depending on which card you are using. The important thing to know is that the “bus” only 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:

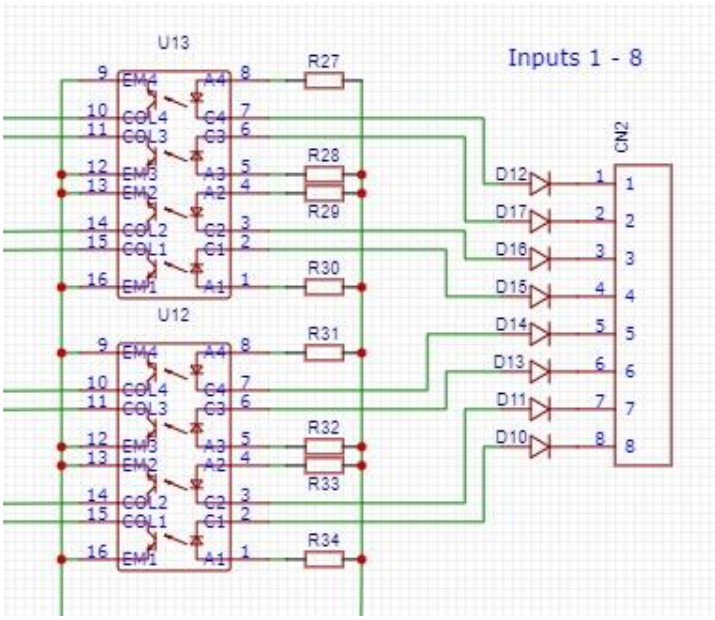


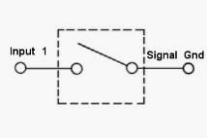
The Signal GND terminal strip (P4) terminals (1) and (2) are jumpered together on the board. Use these terminals to provide the connection to the common side of the field devices to be controlled. You can also use this ground to connect the common side of the input devices.

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:

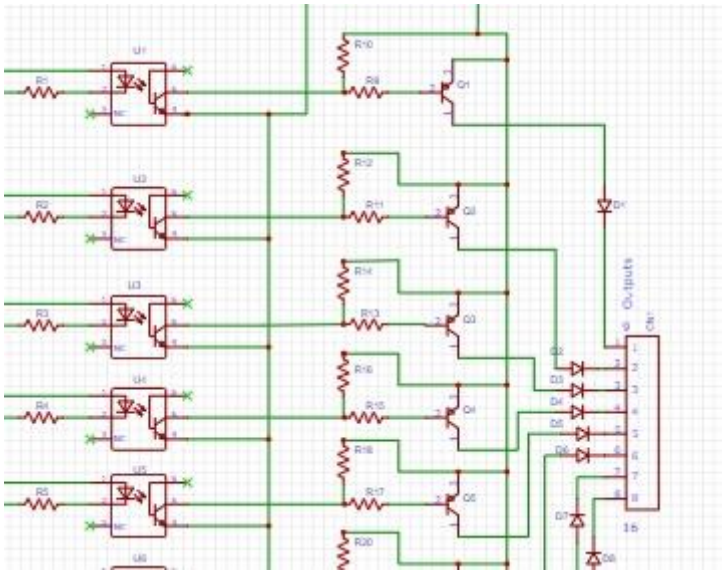
There are eight inputs on the board on terminal block CN2. Input 1 is on terminal (1) and Input 8 is on terminal 8. Examining Input 1 we see diode D12, this is a reverse polarity protection diode. D12 is connected to U13 pin (7) this chip provides galvanic isolation for the Nano input terminals. Resistor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

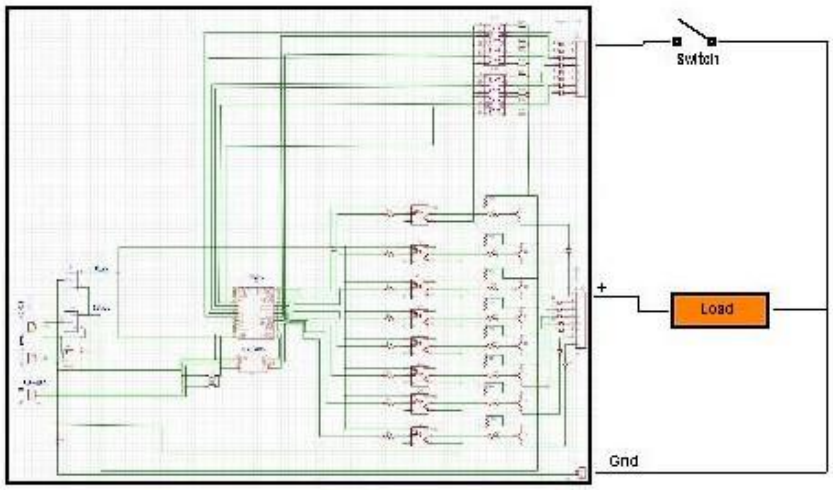




Outputs 1-8:

There are eight outputs on the board on terminal block CN1. Output 1 is on terminal (1) and Input 8 is on terminal 8. Examining Output 1 we see diode D1, this is a reverse polarity protection diode. D1 is connected to the collector of Q1, the power output transistor providing 1 Amp of current. The optocoupler chip (U1) provides galvanic isolation for the Nano output terminals. Q1 acts like a switch passing the +12 VDC from its emitter pin (3) to its collector pin (1) and out to CN1 terminal 1.





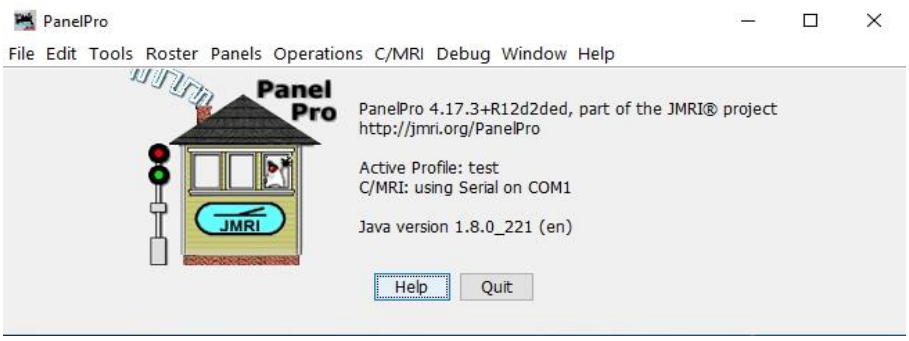
This is an example of a typical input and output connection.

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.

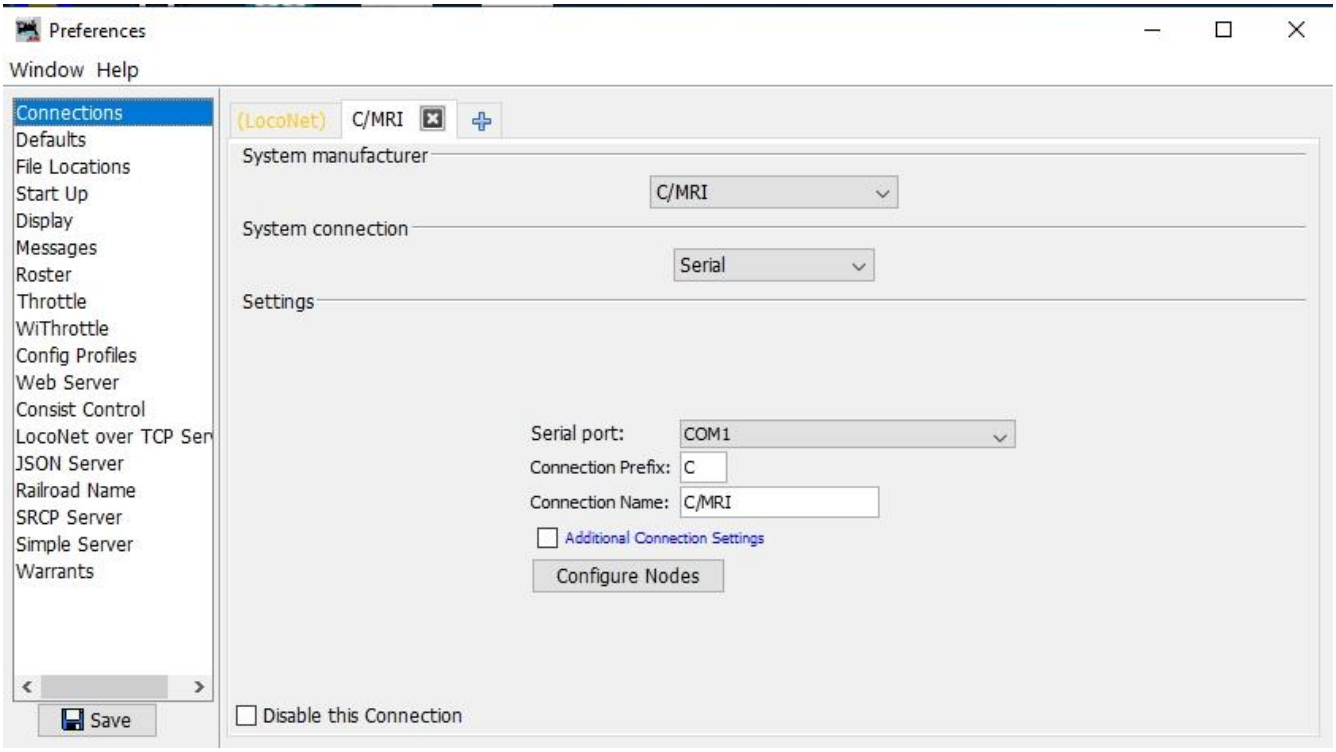
This overview of how to program a C/MRI port is not a comprehensive guide to JMRI but it should be enough information to get you started.



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.



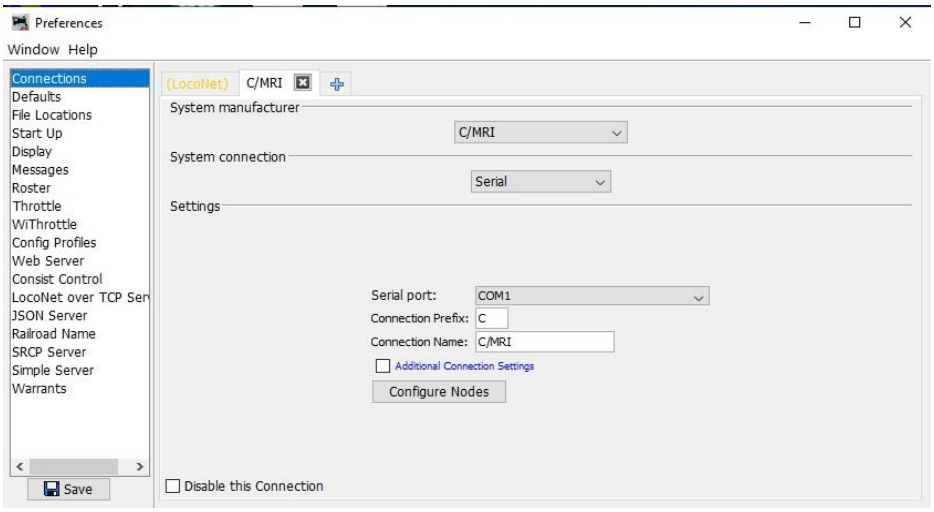
Now select connections from the side-bar menu.
Your window will now have at least two tabs, one is usually named for the DCC system you are using and the other has a big “plus” sign. Click the “plus” sign to add a new connection for JMRI to use.
From the drop-down list boxes select the following:

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

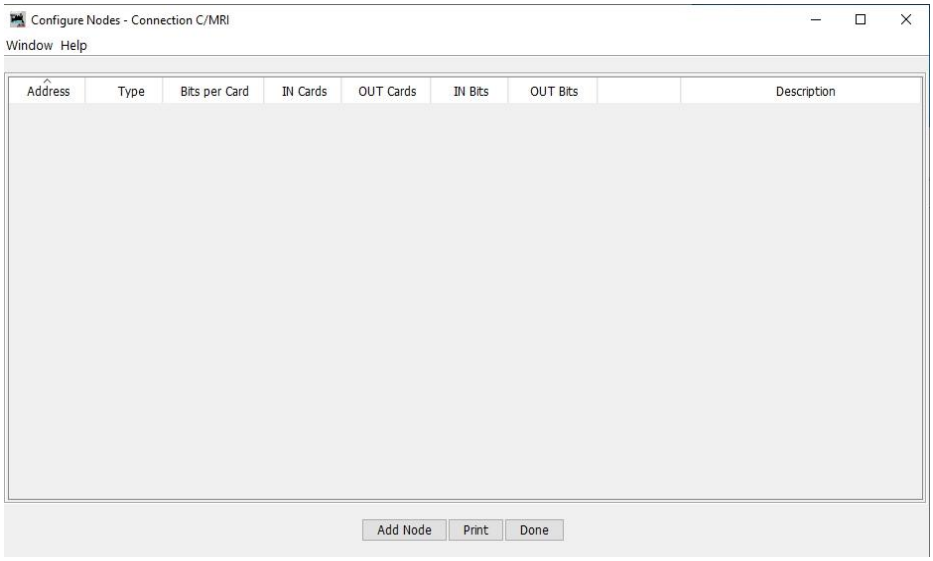
Leave the connection prefix with the default “C.”
The Connection Name can be anything you want it to be, I suggest C/MRI as this is the name that will show up on the tab and in other menu choices making it easy to remember what hardware you are talking too.
Additional Connection Settings: Set the baud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

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.

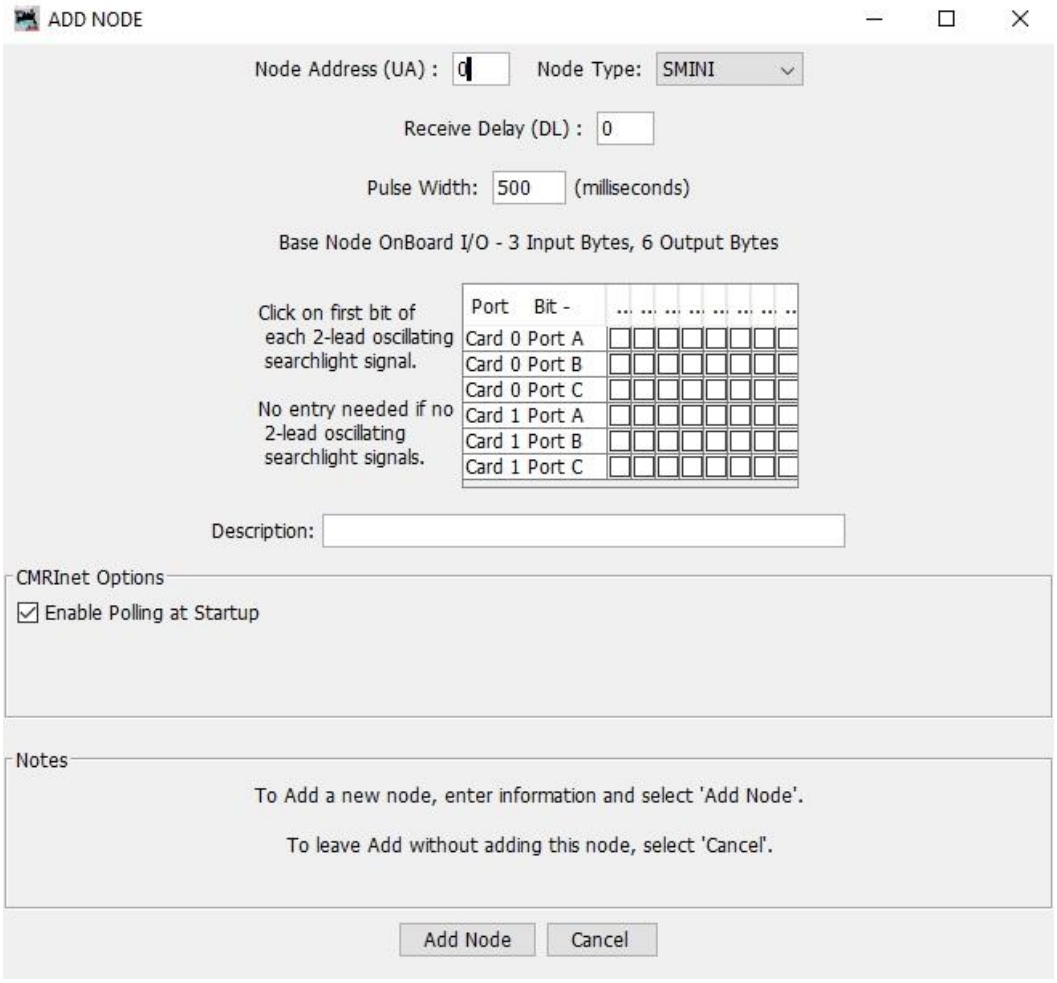
From the splash screen menu go to EDITS/PREFERENCES



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



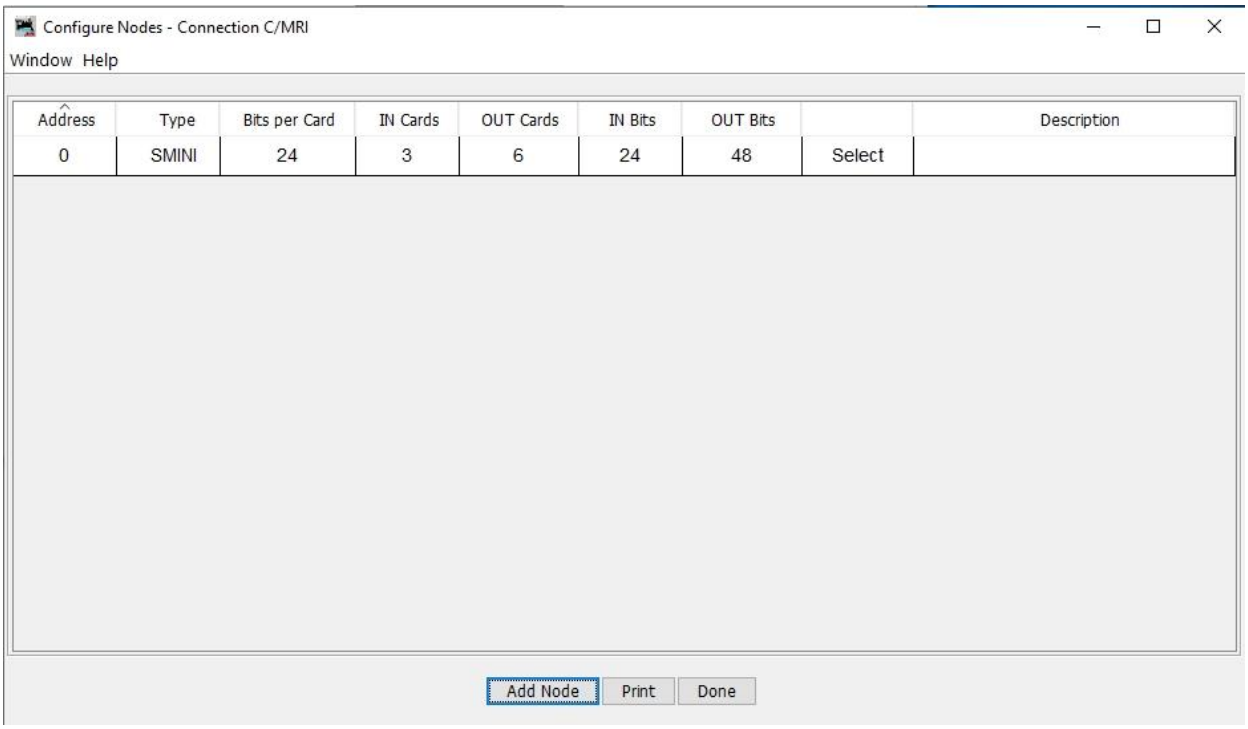
A new blank node list pop-up will be displayed. Click the "Add Node" button to open the next pop-up



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.

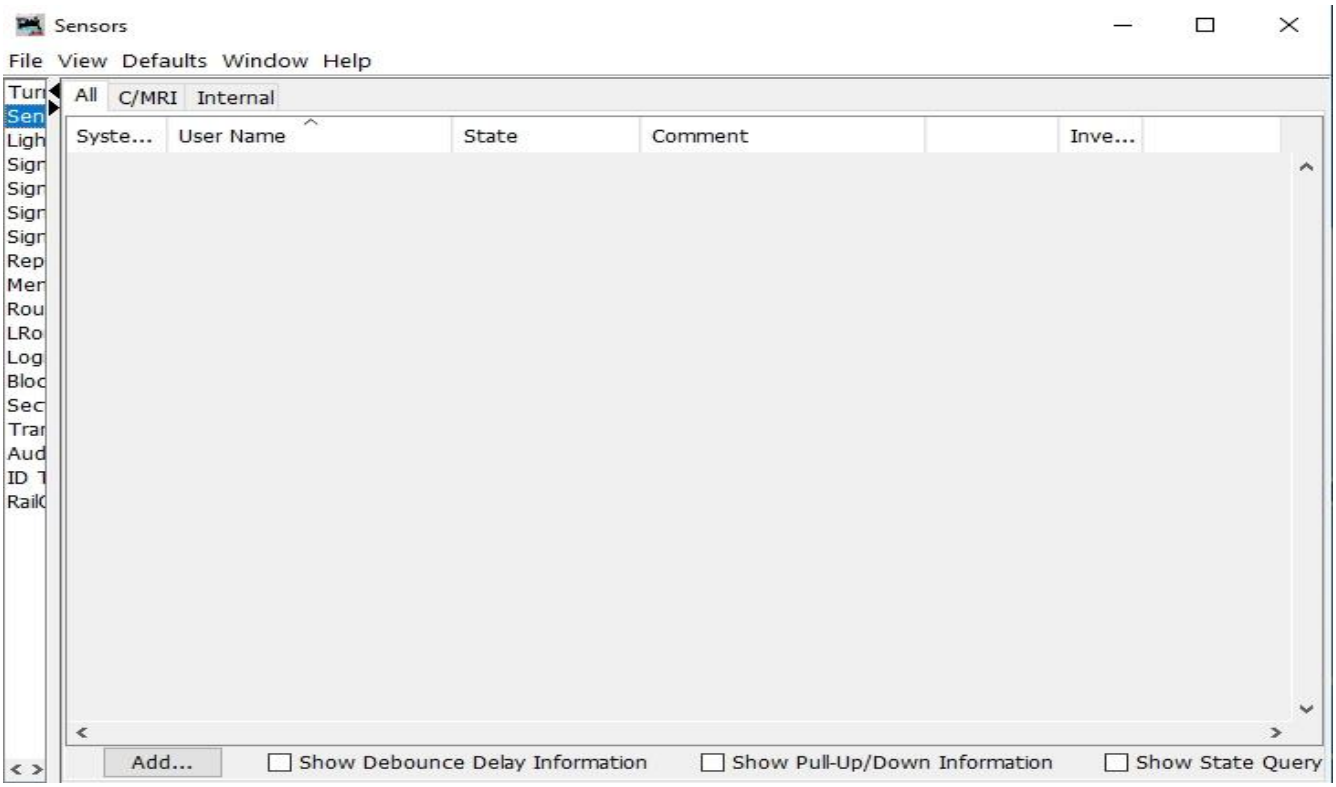
Click to “Add Node” button to close this pop-up



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.

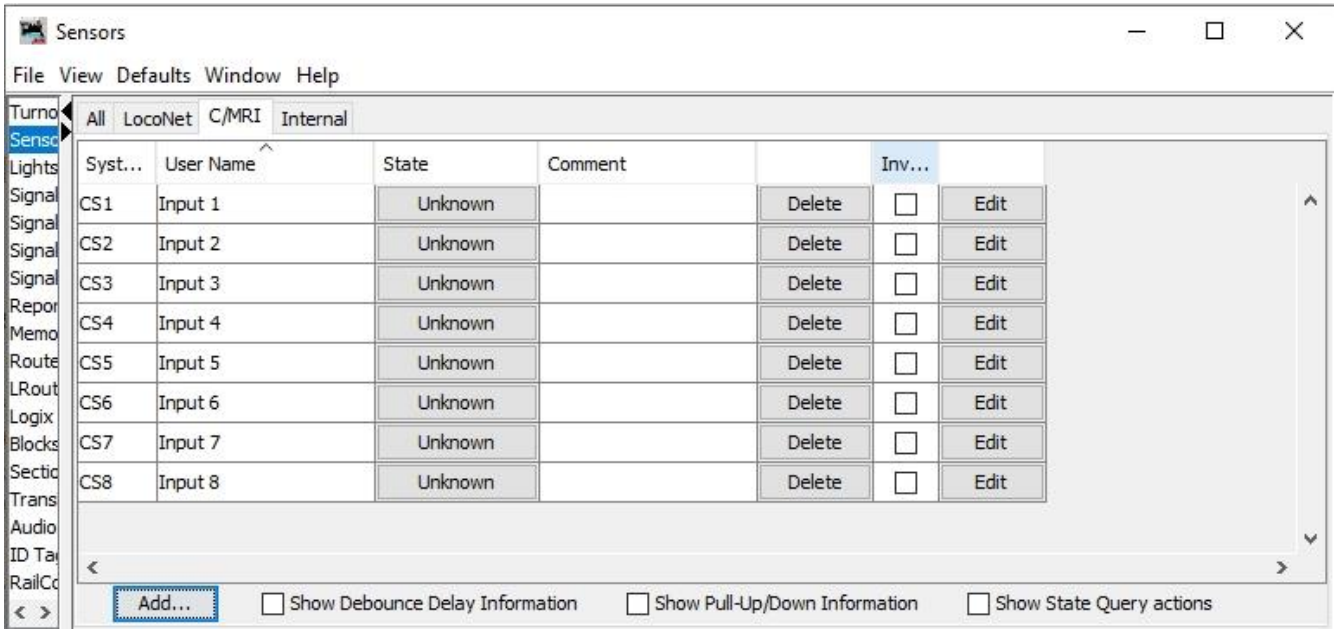
We now have a C/MRI node configured and ready for JMRI to use.

From the splash screen menu select TOOLS/TABLES/SENSORS

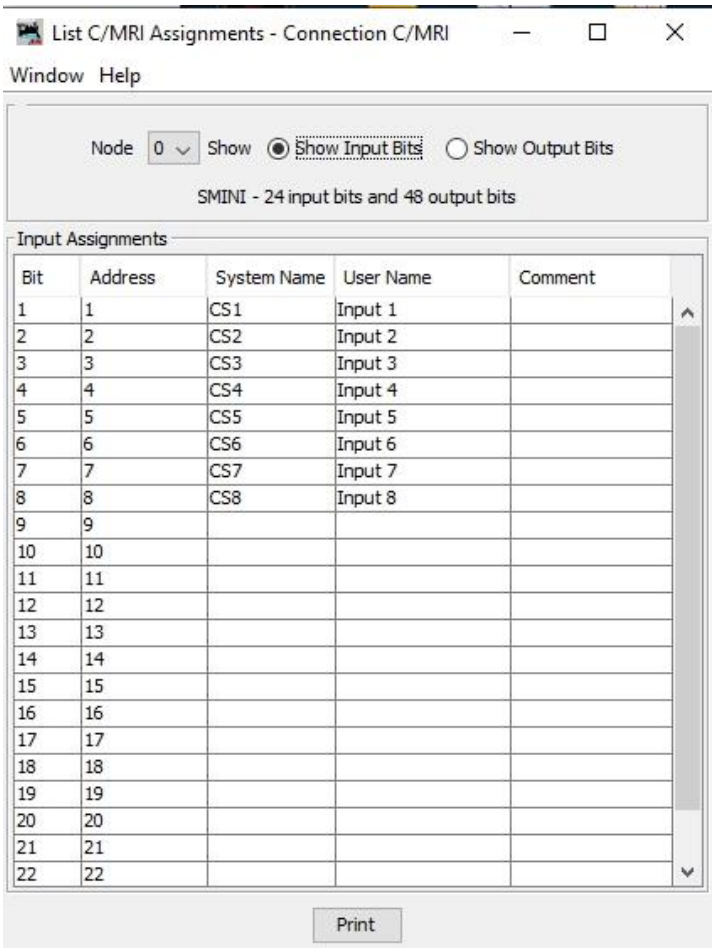


Sensors are input signals for JMRI to use for status displays and logic decisions. We have eight (8) inputs on the card to define. Click the “Add” button to get started.

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.


Sensors
File View Defaults Window Help
All LocoNet C/MRI Internal
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
Add...
Show Debounce Delay Information
Show Pull-Up/Down Information
Show State Query actions

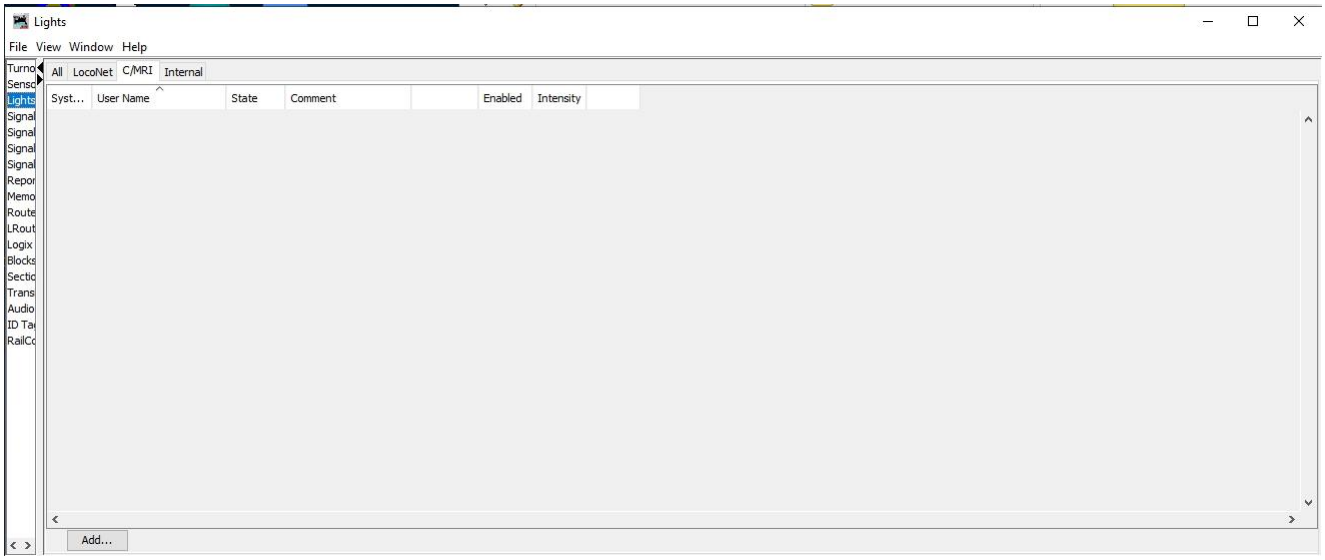
From the Splash Screen got to C/MRI/List Assignments



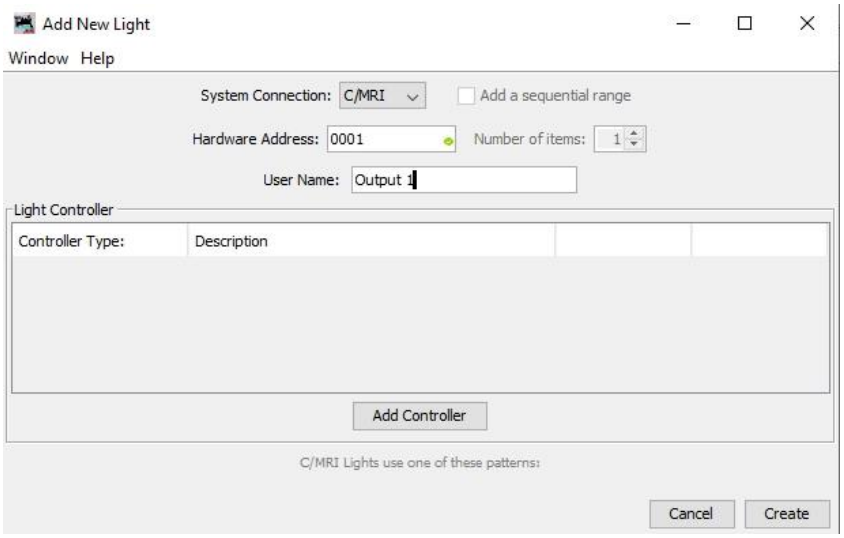
Check that you're on "Node 0"
Click "Show Input Bits"

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

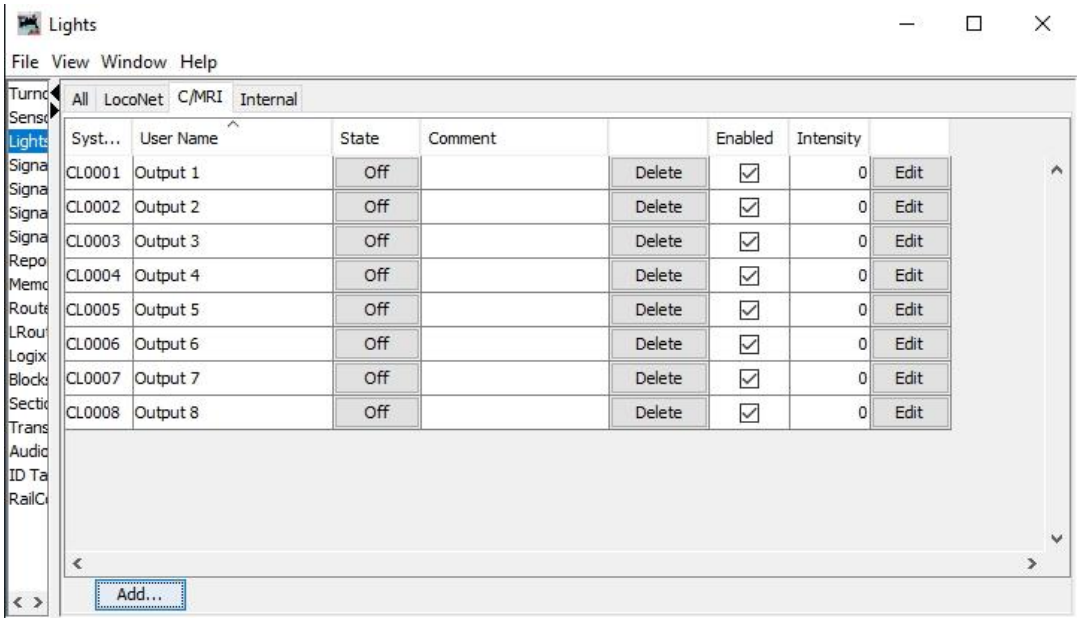
From the splash screen menu select TOOLS/TABLES/LIGHTS



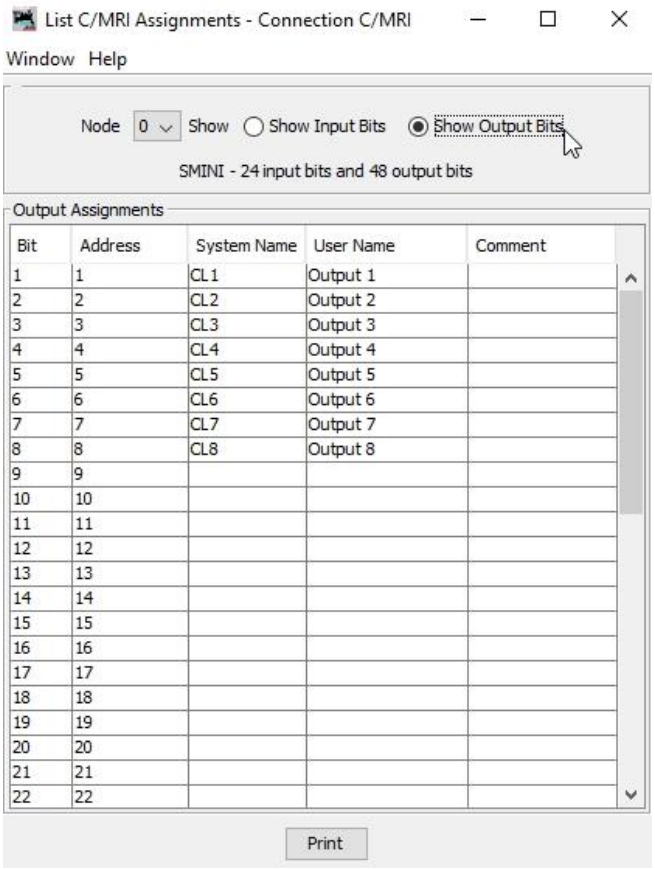
Lights are simple output signals from JMRI to be used to control field devices. We have eight (8) outputs on the card to define. Click the “Add” button to get started.



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 Hardware Addresses will be 0001 – 0008.
We start with address 0001 and give it a User Name of our choosing (“Output 1” is my demo name.) Press the “Create” button and repeat this process to add the other seven inputs as shown below.



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



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.

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.

// Use English words “input_X” to represent the cryptic hard ware addresses “A5” etc

// eight inputs
#define input_1 A5
#define input_2 4
#define input_3 3
#define input_4 A4
#define input_5 A3
#define input_6 A2
#define input_7 A1
#define input_8 A0
```

```
// Use English words “output_X” to represent the cryptic hard ware addresses “12” etc

// eight outputs
#define output_1 12
#define output_2 11
#define output_3 10
#define output_4 9
#define output_5 8
#define output_6 7
#define output_7 6
#define output_8 5

// a dedicated hardware pin for Auto485 code to use to control data flow

#define DE_PIN 2
Auto485 bus(DE_PIN);

// set the node address change “0” to match your node address

#define CMRI_ADDR 0
// create a software OBJECT to store input & output data

CMRI cmri(CMRI_ADDR, 24, 48, bus);

// Do this stuff once before beginning the code

void setup() {
    //  start the Auto 485 communications

    bus.begin(9600,SERIAL_8N2);

    // define these Uno hardware pins as inputs

    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);

   // define these Uno hardware pins as outputs

    pinMode(output_1, OUTPUT);
```

```
    pinMode(output_2, OUTPUT);
    pinMode(output_3, OUTPUT);
    pinMode(output_4, OUTPUT);
    pinMode(output_5, OUTPUT);
    pinMode(output_6, OUTPUT);
    pinMode(output_7, OUTPUT);
    pinMode(output_8, OUTPUT);
}

// loop keeps repeating the code over and over

void loop() {

    // Inputs to C/MRI
// start the code in the imported CMRI library

cmri.process();

// define some variables to remember things and count stuff

    boolean a = LOW;
    boolean b = LOW;
    int offset = 0;

// for means repeat a set number of times
// int i holds the value to test and is set here to 1
// <= 8 compare it to int and as long as it is less than 9 repeat the loop
// i++ add one to int each time the for starts

    for (int i = 1; i <= 8; i++) {


     //   switch means pick a matching case (i) and start from there

       switch (i) {
          case 1:
              // read value in the Nano input and store it in “a”
                  a = digitalRead(input_1);
              // break means jump over the rest of the code to the end of the switch statement
                 break;
          case 2:
             a = digitalRead(input_2);
             break;
          case 3:
             a = digitalRead(input_3);
```

```
            break;
         case 4:
            a = digitalRead(input_4);
            break;
         case 5:
            a = digitalRead(input_5);
            break;
         case 6:
            a = digitalRead(input_6);
            break;
         case 7:
            a = digitalRead(input_7);
            break;
         case 8:
            a = digitalRead(input_8);
            break;
         default:
            // statements
            break;
      }  // end of switch statement
// if/else means IF true do this; ELSE when false to do this
// compare the value stored in “a” and invert that value

      if (a == LOW) {
        b = HIGH;
      } // end of if
      else {
       b = LOW;
      } // end of else
// remember CMRI addressing starts at zero and our “for” starts at 1 so subtract 1 from i

      offset = i-1;
// pass the Uno hardware bit value into the CMRI software which sends the input status to the
computer.

      cmri.set_bit(offset,b);
   } // end of for statement loop back up and repeat

boolean x = LOW;
boolean y = LOW;

// here we get the CMRI software value and pass it to the proper Uno hardware bit.
// Note that we invert the value first (for) and then pass it to the Uno (switch)
*****
// get value for the output from CMRI
```

```
    for (int i = 0; i <= 7; i++) {
     x = cmri.get_bit(i);
       if (x == LOW) {   // invert the logic
          y = HIGH;
       }
       else {
         y = LOW;
       }
       switch (i) {
          case 0:
               // send value to the Nano hardware bit
               digitalWrite(output_1,y);
               break;
          case 1:
             digitalWrite(output_2,y);
             break;
          case 2:
             digitalWrite(output_3,y);
             break;
          case 3:
             digitalWrite(output_4,y);
             break;
          case 4:
             digitalWrite(output_5,y);
             break;
          case 5:
             digitalWrite(output_6,y);
             break;
          case 6:
             digitalWrite(output_7,y);
             break;
          case 7:
             digitalWrite(output_8,y);
             break;
          default:
             // statements
             break;
       }
    }


}
```