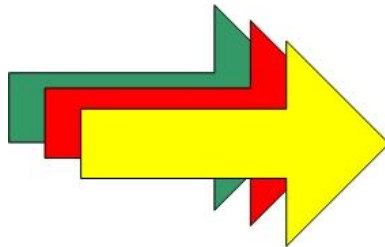


Control Master

A Complete Distributed Control Solution for the PC

Installation and Users Manual
(Including Control Master Software)



Available exclusively from
PC Control Ltd.

www.pc-control.co.uk

© 2009 Copyright PC Control Ltd.

Revision 1.3

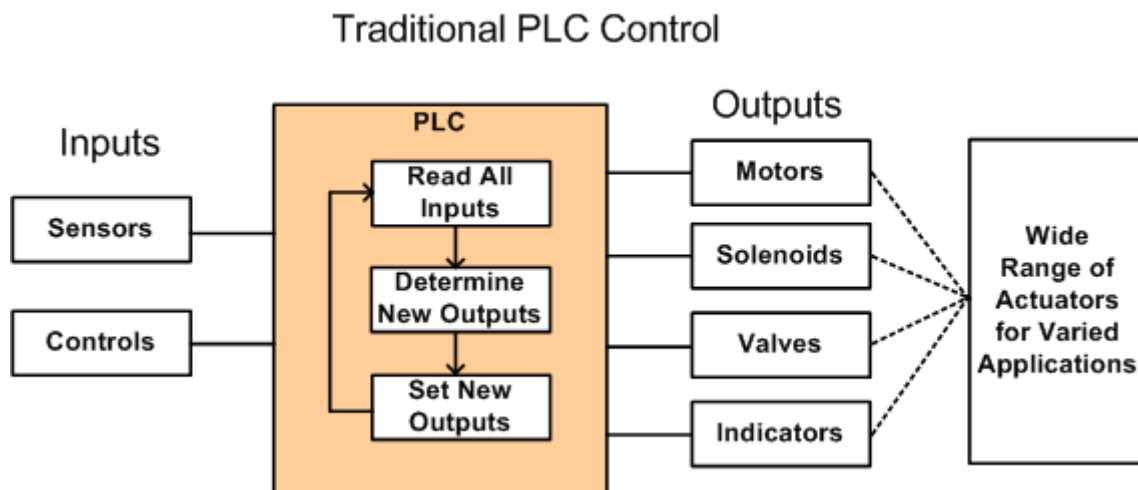
Contents

- 1 Introduction**
 - 1.1 What is Distributed Control
 - 1.2 Control Master Distributed Control System
- 2 Master Controller**
 - 2.1 Installing Master Controller
 - 2.2 Connecting the Master
- 3 Digital Input / Output Slave**
 - 3.1 General
 - 3.2 Board Numbering
 - 3.3 Inputs
 - 3.4 Outputs
 - 3.5 Digital I/O Slave Connector Pinouts
- 4 Analogue Input / Output Slave**
 - 4.1 General
 - 4.2 Board Numbering
 - 4.3 Inputs
 - 4.4 Outputs
 - 4.5 Analogue I/O Slave Connector Pinouts
- 5 Relay Slave**
 - 5.1 General
 - 5.2 Board Numbering
 - 5.3 Inputs
 - 5.4 Relay Outputs
 - 5.5 Relay Slave Connector Pinouts
- 6 Motor Slave**
 - 6.1 General
 - 6.2 Board Numbering
 - 6.3 Outputs
 - 6.4 inputs
 - 6.5 Motor Slave Connector Pinouts
- 7 Control Master Software**
 - 7.1 Common Controls
 - 7.2 Digital Input / Output Page
 - 7.3 Relay Page
 - 7.4 Analogue Input / Output Page
 - 7.5 Motor Page
- 8 Programmers Guide**
 - 8.1 DLL Functions Reference
- 9 Minimum PC System Requirements**

1. Introduction

1.1 Distributed Control: What is distributed control ?

An electrical, electronic or electromechanical system where a number of devices are “controlled” in some way based on the requirement to perform a pre-specified set of operations while also , possibly, taking account of sensor readings, can be described loosely as a control system. In any control system there is a controller which acts as the “brain” behind the process. In a typical PLC control system the PLC controller is a single entity with lots of inputs and outputs suited to a wide variety of devices and sensors.



Once the PLC has been programmed with the particular requirements of the application and it has been connected to all its inputs and outputs , it is ready to control. The PLC would typically repeat the following loop continuously as it performs its tasks....

1. Read all Inputs
2. Determine New Outputs based on current state and current inputs
3. Set the New Outputs

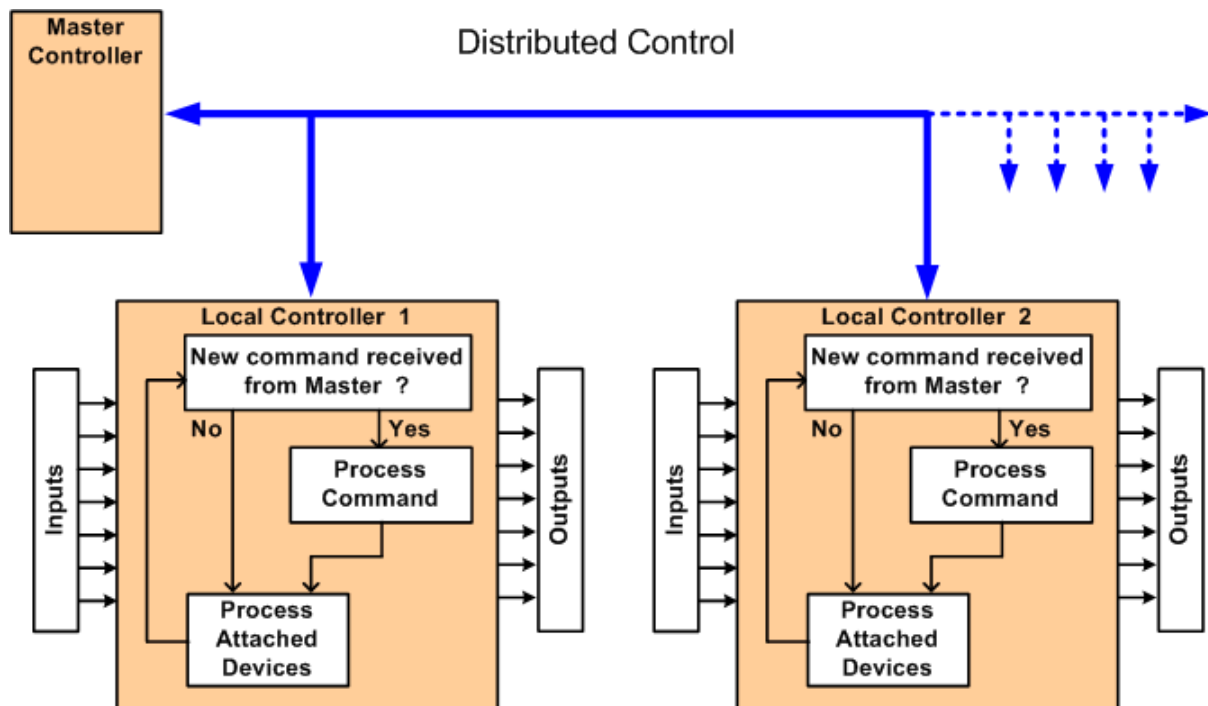
All processing needed to be done in carrying out this loop would be done by the PLC’s own local processor. All wiring to and from the sensors and controlled devices would also be routed to somewhere near the PLC’s processor. Typically this would be in a cabinet fitted with DIN rails holding the PLC module and additional modules for connection to the “devices”.

In contrast to this is the idea of distributed control. In this case there is not just one processor dealing with all of the system. The overall system is divided up into local processing modules which can read and set their own attached devices/sensors. Each different module can specialise in a particular type of device: for example, motor control, digital input/output, analogue measurement, relay switching etc... etc... However, each module must have some general guiding control which coordinates the operation of all of the modules in a given system. This means that there must be communication between them.

This can take many forms but some of the simplest and most reliable are those using serial multi-drop protocols such as RS485. This has the convenience of requiring just two wires to link together all modules in a multi-module system. It should be noted that the ability to have separate modules linked only by a pair of wires can greatly reduce the overall wiring effort in a control system since the modules can be placed near to their own devices. The “guiding control” can be provided by a “Master Controller” which has the ability to communicate with all of the modules in such a system. Although it may seem that taking this approach to distributed control has just added an extra layer of complexity to a simple control system, it is the separation of the “housekeeping “ processing to each of the

individual module processors that makes it very efficient and generally easier to program and maintain.

For example: If the PLC processor needed to make a measurement of a temperature using one of its attached sensors it would need to initiate the analogue to digital conversion of the “raw” input signal before using the result in its main job of deciding what to do at “such and such” a temperature. When doing the same thing using a dedicated analogue input module the “Master Controller” can simply request the current temperature from the module using the comms channel. The analogue module would carry out the A/D conversion automatically in the background. Similarly, a stepper motor requires a special sequence of pulses to each of its phase coils to make it move in a particular direction at a particular speed. With a separate stepper motor controller it would simply require the “master” to send a command telling it to take “so many steps” at “such and such a speed” leaving the modules local processor to take care of the correct pulse sequence generation.



Another advantage in distributed control is the ability to have the modules separated by quite large distances without having to worry about cable lengths and associated power distribution problems. For example: To operate a large motor requiring large currents at a long distance from the controller (eg 1Km) would require expensive cable to prevent voltage drops between the supply and the motor. The local controller option means that the motor power supply can be situated exactly where it is needed; close to the motor and controller.

1.2 Control Master Distributed Control System

The Control Master system by PC Control Ltd. offers an easy to use solution for distributed control. Its main features are....

- USB 2.0/1.0 Master Controller with optional 2500v isolation facility
- Just two wire link between Master and, up to 30 Slave modules
- Distance to farthest Slave can be a massive 1Km
- An ever increasing range of Slave module types to cover most applications
- Control Master Software for instant control of all attached devices
- A DLL (Dynamic Link Library) for programmers to write their own programs to access all of the attached slave modules (compatible with most languages C, C++, VB etc)
- Compatible with Windows XP and Vista
- The lowest cost control option of its type available

Each of the elements that go together to make up the “Control Master Distributed Control System” will now be described in detail showing how to install, connect and operate them.

2. Master Controller

Control Master uses a Master–Slave principle of operation. In all systems there is one “Master Controller” board which is connected to a computer via a standard USB lead and up to 30 slave boards.



The job of the Master is to allow the PC to send and receive data to any of the slave boards. The Master has an on board Microcontroller which is dedicated to performing two main functions..

1. Handling the USB interface to the PC
2. Handling the RS485 serial communications to the slave boards

There are two different types of Master Controller available. The “Non-isolated Master” is the simplest form offering a direct connection between the PC the Master and the Slave boards. In this configuration all of the boards share the same ground connection as the PC. In many applications this is perfectly acceptable. However, when there is a requirement to control an “electrically noisy” device which can generate large voltage transients it can be safer to separate this electrically from the PC. The “Isolated Master” achieves this by using high speed opto-isolation components that removes the direct connection between Master and Slave and replace it with an optical signal. This is actually a light emitting diode and a light sensor encapsulated in the same small 8-pin chip. This “light connection” provides isolation for up to 2500 volts. In other words, if there are voltage spikes up to 2500 volts on any of the slaves, they are prevented from reaching the sensitive components of the PC. Installing, connecting and operating the Master Controller is exactly the same whichever version you use.

2.1 Installing Master Controller

Before connecting the Master to a USB port it is necessary to install the “Control Master” software from the installation CD. The software is compatible with **Windows XP** and **Vista** only. To begin the installation, insert the disk into your CD drive. The installation program should start automatically, but, if it does not then use windows explorer to navigate to the disc and double click on the “Setup.exe” file. Installation is very quick and simple; just follow the on screen instructions. Note that all installation procedures described should be done having logged on as Administrator with full administrator privileges. Also note that use of the driver is restricted to those users with Admin privileges and you should therefore login to windows as administrator whenever actually using this software and associated drivers.

Once installed you should connect the Master Controller to a free USB port. Although the Master is a USB 2.0 device it can be used on a USB 1.0 port with no loss in performance. Windows should acknowledge the connection of the Master by an audio signal

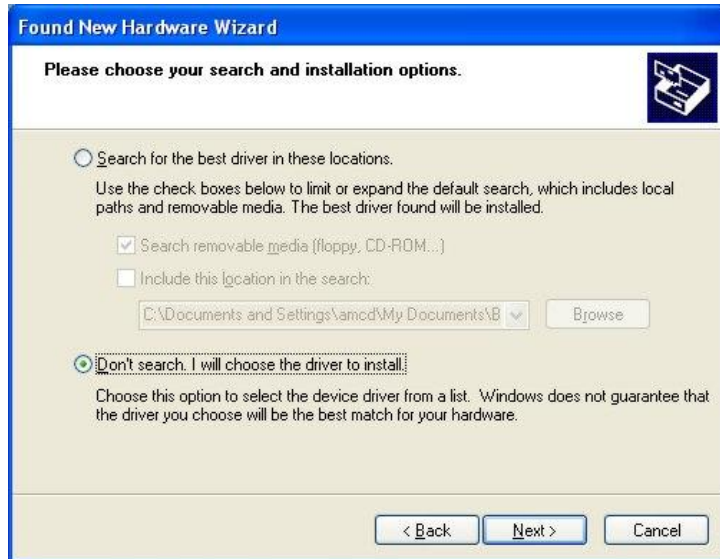
(usually two rising tones) and a prompt to say that it has found new hardware with the following window...



Select "No, not this time" and click "Next". You will then see...



Select "Install from a list or specific location (Advanced)" and click "Next" which will lead to the screen below...



Choose "Don't search. I will choose the best driver to install" and then click "Next"



When the above screen appears click on the "Have Disk" button. Your installation CD should be in your CD drive at this time.



When presented with the above screen, make sure the drive selected "eg D:\\" is the correct one on your system that has the installation CD in it. Click "OK"



The wizard should find the “PC Control Board” and clicking next should begin the installation. If you are presented with the following box...



... simply click “Continue Anyway”.

The installation should then complete automatically with a final screen showing..



.. which requires you to click on Finish. You may also see an information bubble appear informing you that your new hardware is “installed and ready for use”.

Windows 7 Additional Notes for Windows 7 Installations:

When the board is first connected to a USB port windows may try to automatically install a driver for it. This will fail and windows will inform you of this. Simply close the message box and proceed to install the driver manually as follows...

Go to the control panel and click on "View devices and printers".

In the list of devices you will see the attached board. Double click on this and choose the "hardware" tab. Then choose the properties to see the device properties dialog box.

Click on the "Driver" tab and choose "Update Driver".

Choose “Browse my computer for driver software” and then select the drive which contains the installation CD (followed by "Next").

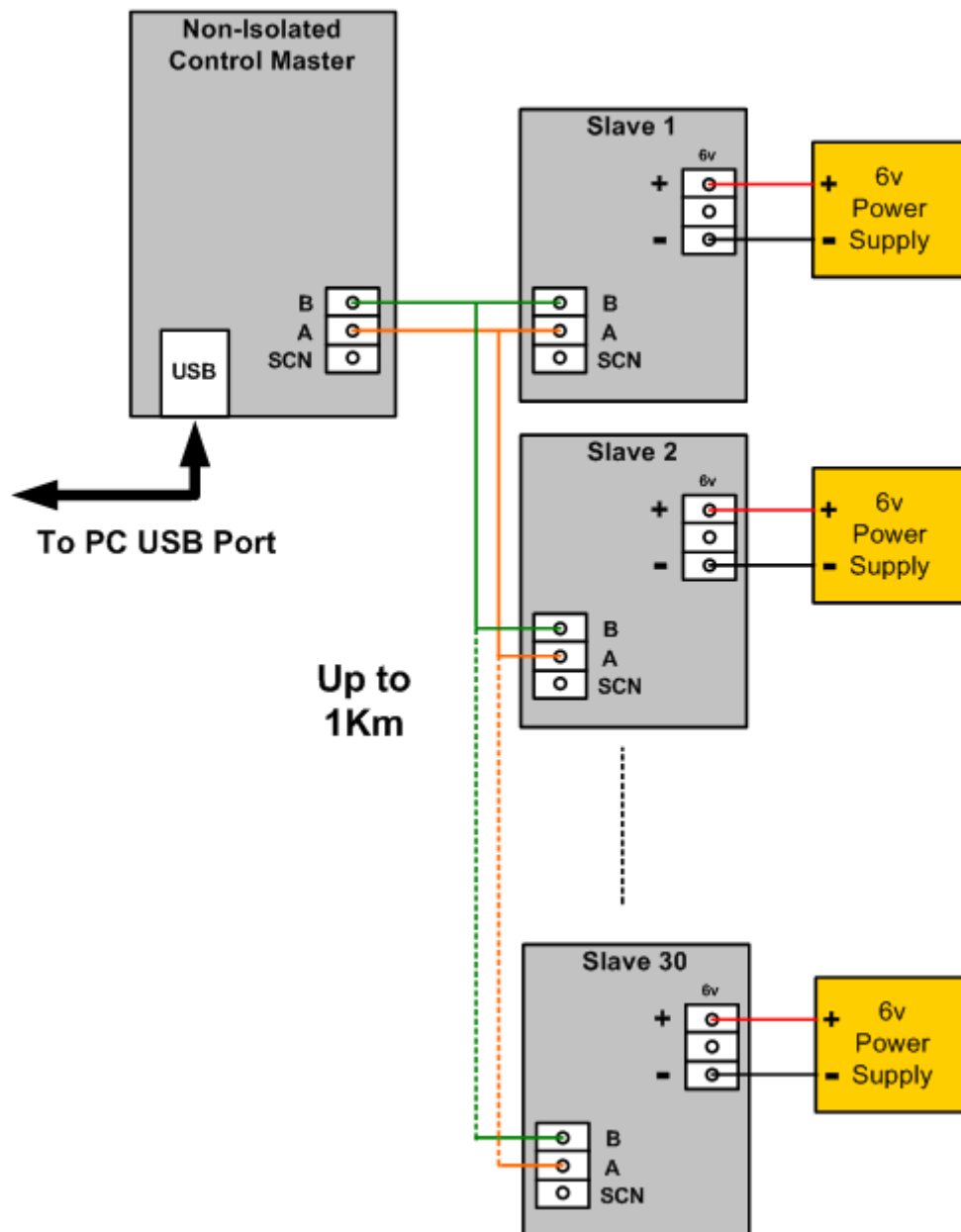
Windows should then install the driver and confirm it has done so..

The board is now ready to be used with its supplied application software or your own programs using the DLL supplied.

2.2 Connecting the Master

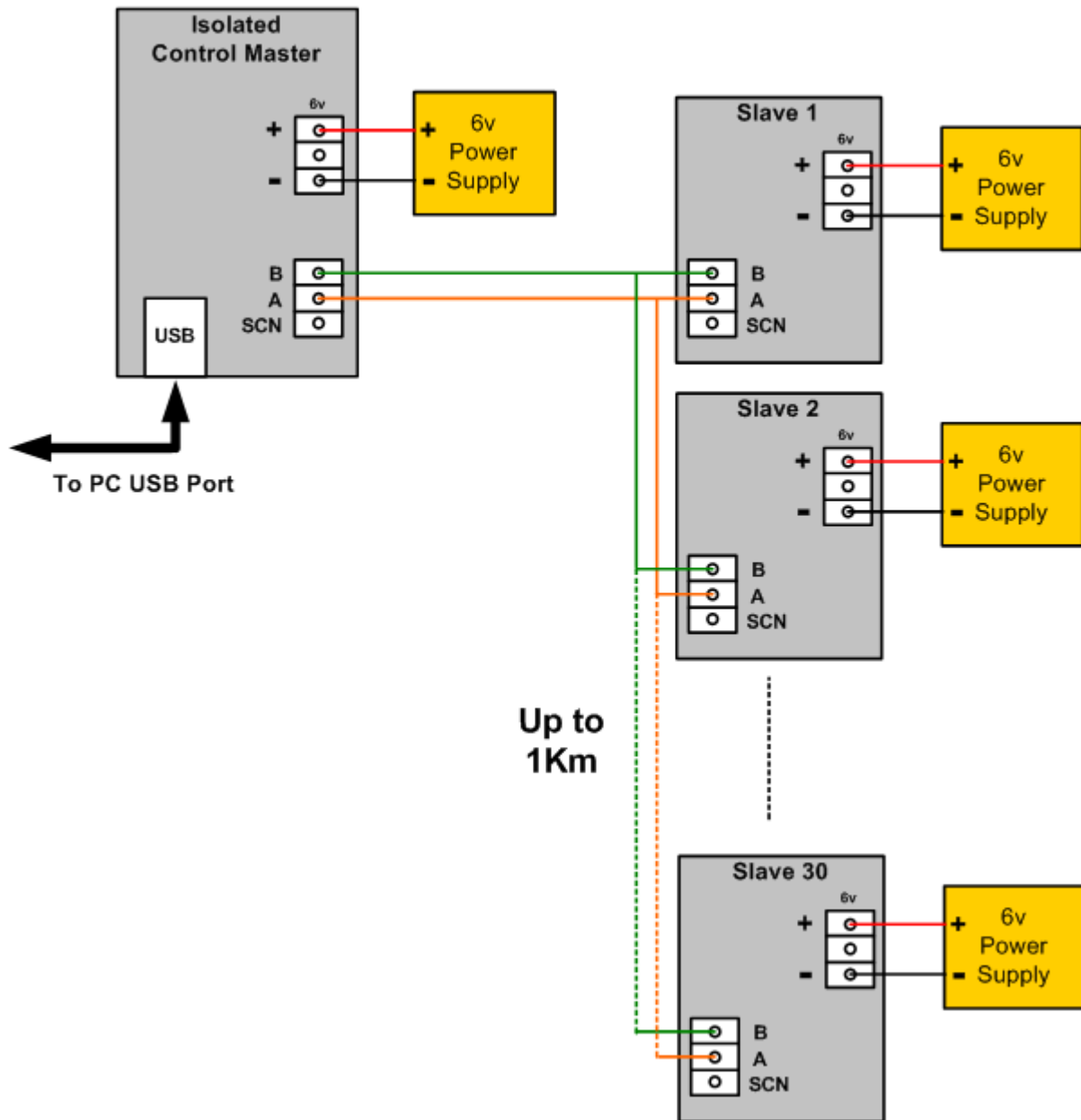
One of the few differences between the Isolated Master and the Non-Isolated version is the power supply requirements. The **Non-Isolated** master becomes fully operational as soon as you connect it to a USB port. i.e. it receives all of its power from the USB port.

Connecting Slaves to a Non-Isolated Master



However, the **Isolated** Master is only partly powered from the USB port. The Microcontroller and associated components on the Master are powered from USB whilst the RS485 communications components require a separate supply. This should be provided on the terminals marked 6v (+ and -). Although it is labelled as 6v, the external power supply can be anything from 6v minimum up to 12v maximum. The Master has an on-board regulator designed to accommodate this range of input voltages. Whatever supply voltage is used, it should be fully regulated and a good quality DC supply capable of at least 100mA.

Connecting Slaves to an Isolated Master



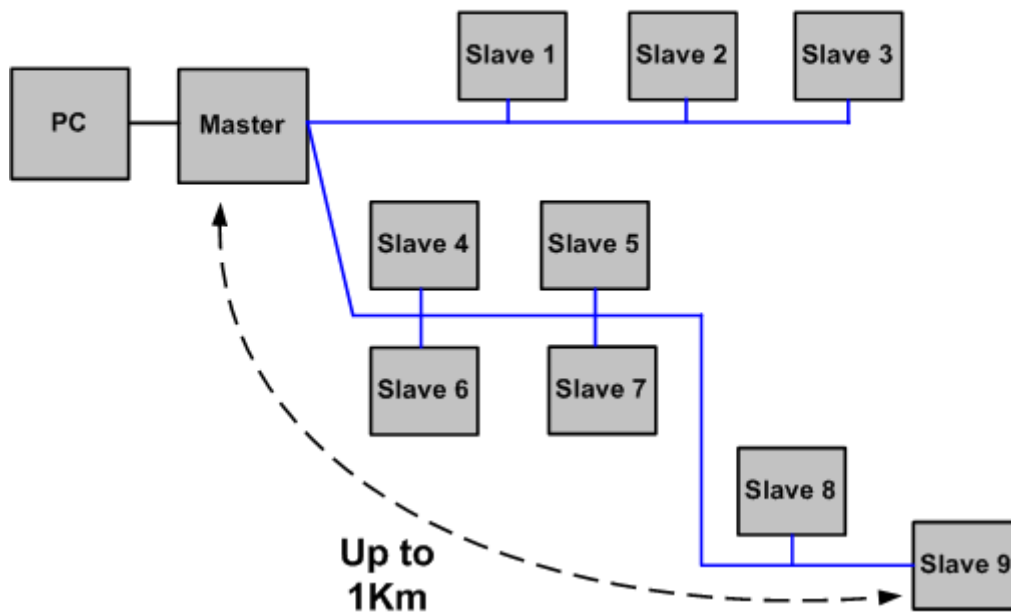
The reason that the Isolated Master requires an additional supply for its RS485 chips is to maintain the electrical isolation between the PC and the external circuitry. i.e. if you were to use one of the PC's power supplies for providing this voltage you would have effectively bypassed the isolation provisions. You should give thought to your overall connection strategy at this point if you are using an Isolated Master. All of your external slave devices should be powered from supplies not linked in any way with the PC to maintain isolation (including the above mentioned power to the Master). It should be noted that there is no reason why the one 6v power supply could not be used to power the isolated master and all of the slaves as long as that power supply was capable of supplying enough current to power all boards attached. Note: There are some advantages is using 4 core cable rather than the single twisted pair which normally links the slaves to the master. The additional two cores could be used to carry the 6v supply to all slaves.

Once fully powered, the Master simply connects to a slave by connecting two wires. These wires are connected to terminals TL1 and are labelled A and B. Connect the A on the

Master to the A on the slave and similarly B to B. This wire can be up to 1Km long and should be twisted wherever possible. The signalling which takes place over this pair of wires is standard RS485 signals which are differential in nature. This means that they are highly immune to electrical noise coming from the surroundings where the cable is routed. Twisting the wires together improves this immunity by making it less likely that an external electrical signal will cause any differential interference voltage in the wires. Connecting to more than one slave is simply a continuation of the two wire connection. This can either be a separate two wires starting at the master and going, perhaps in a different direction, to the second slave or it can be a connection from slave one to slave two. In either case always ensure you connect terminals A to A and B to B. The flexibility in forming a continuous “chain” of slaves connected together and to the Master or to connect them directly to the Master in a “star” configuration allows your wiring to reflect the most suitable routing for your own application.

Also, remember that each slave can also be connected to other slaves in a star configuration. The only limitation is that no slave can be any more than 1Km from the Master in terms of total length of shortest cable route. This flexibility makes the distributed control system very suitable to a wide variety of installations.

Very Flexible Connection Strategies



In particularly noisy (electrically) environments, such as heavy industrial applications, it may be advisable to use additional shielding for the pair of wires. In this case you can purchase commonly available screened twisted pair cable which has the two wires plus an overall metal foil screen, which is in turn covered in a plastic outer layer. When using this type of cable the two wires should be connected as normal and the metal foil screen should be connected to the terminal marked “SCN” on TL1 and on each slave.

Before discussing the Control Master software, the next few sections will explain the details of operation of each type of slave modules in turn. If your system does not use all of the available module types you may want to skip some sections and concentrate on your particular modules and then look at the software that controls them.

3. Digital Input / Output Slave

The Digital Input/Output slave has 8 digital inputs and 8 switching outputs.



3.1 General:

To operate correctly the Digital I/O Slave needs to be supplied with an operating DC voltage of between 6v and 12v. This should be connected to the terminals labelled 6v (+ and -) on TL2. The power supply should be fully regulated and capable of providing at least 100mA. The Board also needs to be connected to a Master Controller using the two wires on TL1 (A and B). Connecting 'A' to 'A' and 'B' to 'B'. Alternatively it can be connected to any other slave module which is already connected to the Master using the same connection strategy. Use of the SCN connection is optional but where used it should be connected to the metal foil shielding on a twisted pair cable.

3.2 Board Numbering:

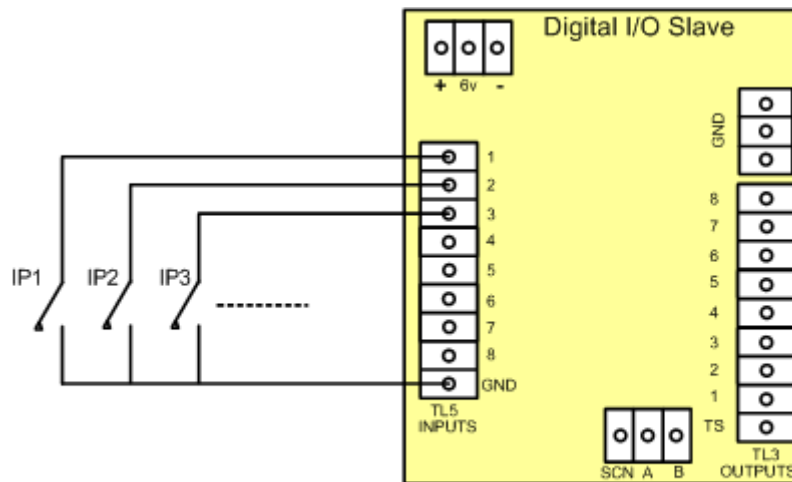
One last task is required before the Digital I/O Slave can take part in the main control system and that is to allocate it a board number. It is necessary to allocate each board a unique "Board Number" so that commands and data from the Master Controller can be directed at the correct slave board. This is done by setting the blue DIP switches on the board labelled "Board Number". Switches 1 to 5 are used for this purpose. Switches 6,7 and 8 are not used. Each switch is either ON or OFF as indicated by the "ON" label on one side. To specify a particular board number you need to choose which switches are on and which off according to the table below.

Board Number	DIL Switch 1	DIL Switch 2	DIL Switch 3	DIL Switch 4	DIL Switch 5
1	OFF	ON	ON	ON	ON
2	ON	OFF	ON	ON	ON
3	OFF	OFF	ON	ON	ON
4	ON	ON	OFF	ON	ON
5	OFF	ON	OFF	ON	ON
6	ON	OFF	OFF	ON	ON
7	OFF	OFF	OFF	ON	ON
8	ON	ON	ON	OFF	ON
9	OFF	ON	ON	OFF	ON
10	ON	OFF	ON	OFF	ON
11	OFF	OFF	ON	OFF	ON
12	ON	ON	OFF	OFF	ON
13	OFF	ON	OFF	OFF	ON
14	ON	OFF	OFF	OFF	ON
15	OFF	OFF	OFF	OFF	ON
16	ON	ON	ON	ON	OFF
17	OFF	ON	ON	ON	OFF
18	ON	OFF	ON	ON	OFF
19	OFF	OFF	ON	ON	OFF
20	ON	ON	OFF	ON	OFF
21	OFF	ON	OFF	ON	OFF
22	ON	OFF	OFF	ON	OFF
23	OFF	OFF	OFF	ON	OFF
24	ON	ON	ON	OFF	OFF
25	OFF	ON	ON	OFF	OFF
26	ON	OFF	ON	OFF	OFF
27	OFF	OFF	ON	OFF	OFF
28	ON	ON	OFF	OFF	OFF
29	OFF	ON	OFF	OFF	OFF
30	ON	OFF	OFF	OFF	OFF

The number of the board can be any of the available numbers (1 – 30) but make sure you do not have two boards with the same number. The software which will control this board (discussed later) will use this number to specify the particular board.

3.3 Inputs:

The 8 digital inputs have characteristics compatible with standard 5v logic devices. i.e. when the input is at +5v it will be read as a logic '1' or "High". When the input is at 0v or GND it will be read as logic '0' or "Low". As a convenience for use in control systems, the inputs also have "on board" pull ups. This is simply a 10K resistor connected between each of the inputs and the board +5v supply. This means that any unused inputs can be left disconnected without concern over what voltage they are "floating" to. Since the inputs are very high impedance (cmos type) they would "float" somewhere between 0 and +5v if left unconnected giving inconsistent results when read. Unconnected inputs would therefore be read as a logic '1'. The 10K resistor is sufficiently high value so as not to impose a great load on any signal source connected to it and also makes it very easy to use switches on the inputs. i.e. a switch can be connected directly between any input and the 0v (GND) connection to provide a functional digital input. When closed the input would read as logic '0', and when open as logic '1'.

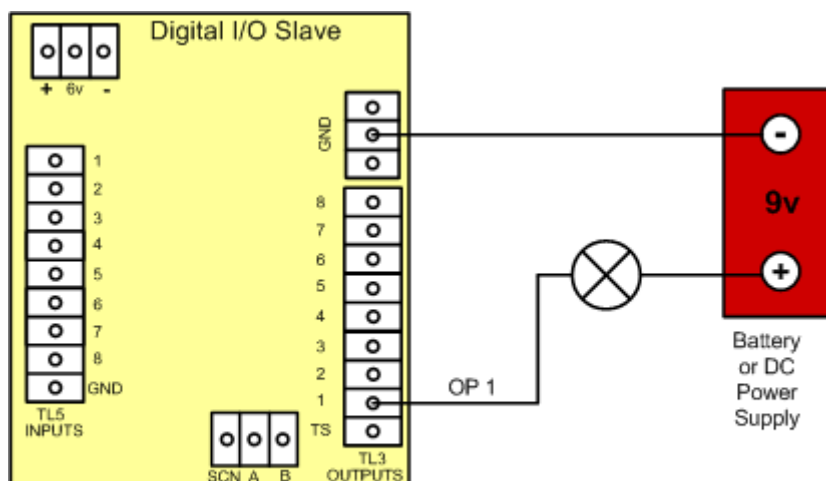


Connecting Switches as Inputs to Digital I/O Slave

3.4 Outputs:

The switching outputs are slightly different to the digital inputs in their characteristics. They are “open collector” type. This means that they behave like a switch which is either open or connected to 0v. When an output is set to “On” (logic ‘1’) the switching output is closed. If you need to have a straight forward digital signal from one of these outputs, all that is necessary is to connect a resistor between the output and the positive supply of your connected device. The positive supply can be anywhere between 5v and 50v. The choice of the resistor should reflect the input characteristics of your attached device but a typical choice for a 5v system would be around 4k7.

The benefit of having open collector type outputs is that they can also be used for higher voltage and higher current switching than is typical with digital logic. For example each output on the Digital I/O Slave can switch up to 50v at 500mA. This opens the possibilities of connecting a much wider range of electrical and electronic devices directly to the outputs. These can include DC motors, solenoids, lamps, LED’s, Relays etc... etc...

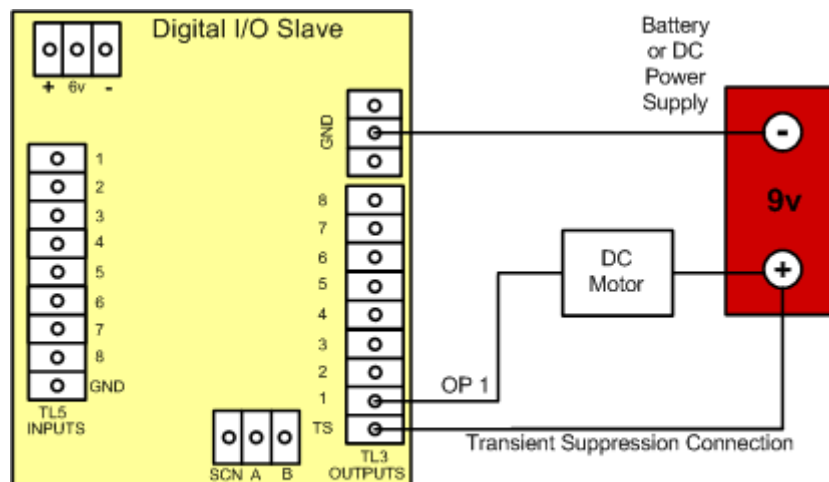


Connecting a Lamp to Digital I/O Slave

By virtue of the fact you can directly drive relays from these outputs, it means that devices with much higher power requirements can be driven from the controlled relay.

When connecting devices with an inductive load (eg motors, solenoids, relays) it is advisable to make use of the transient suppression facility provided on the board. This is simply a series of transient suppression diodes built into the main output driver chip which

act in a way as to “clamp” any transients to the positive external supply. Transient voltage spikes are a result of switching off inductive devices. The collapsing magnetic field acts in a way so as to generate much higher voltages than is normally present. These spikes can be potentially damaging to any connected components and can also cause interference to nearby RF sensitive devices such as a radio. To make use of the suppression facility, simply connect the transient suppression pin directly to the positive terminal of the external supply.



Connecting a DC Motor to Digital I/O Slave

3.5 Digital I/O Slave Connector Pinouts

Pinout of the Digital Inputs On Screw Terminals(TL5)

Pin	Signal description
1	Digital Input 1
2	Digital Input 2
3	Digital Input 3
4	Digital Input 4
5	Digital Input 5
6	Digital Input 6
7	Digital Input 7
8	Digital Input 8
GND	GND (0v)

Pinout of the High Voltage Switching Outputs On Screw Terminals(TL3)

Pin	Signal description
TS	Transient Suppression
1	Switching Output 1
2	Switching Output 2
3	Switching Output 3
4	Switching Output 4
5	Switching Output 5
6	Switching Output 6
7	Switching Output 7
8	Switching Output 8

4. Analogue Input / Output Slave

The Analogue I/O Slave has 7 analogue inputs, 1 digital input and 8 switching outputs. .



4.1 General:

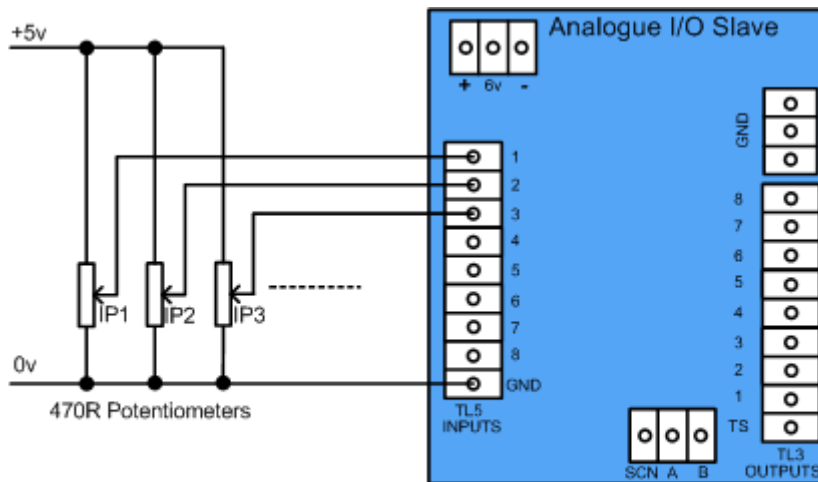
To operate correctly the Analogue I/O Slave needs to be supplied with an operating DC voltage of between 6v and 12v. This should be connected to the terminals labelled 6v (+ and -) on TL2. The power supply should be fully regulated and capable of providing at least 100mA. The Board also needs to be connected to a Master Controller using the two wires on TL1 (A and B). Connecting 'A' to 'A' and 'B' to 'B'. Alternatively it can be connected to any other slave module which is already connected to the Master using the same connection strategy. Use of the SCN connection is optional but where used it should be connected to the metal foil shielding on a twisted pair cable.

4.2 Board Numbering:

The board number has to be set using the blue DIL switches in exactly the same way as already described in the board numbering section of the Digital I/O Slave. See

4.3 Inputs:

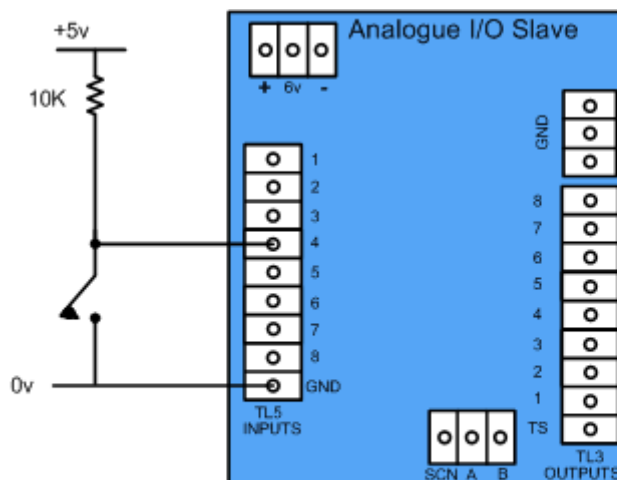
The analogue inputs are designed to work over two ranges which are selectable in software. These are 0 – 5v and 0 – 2.4v. Whenever a voltage appears on an input within the selected range it can be converted to a number in the range 1 – 1023 by the on board A/D (analogue to digital) convertor. The A/D convertor is a 10bit convertor (hence the range 1-1023) giving a very fine resolution of the measurement of the applied voltage. When the range is set to 2.4v the number 1023 corresponds to 2.4v. This means that the measurement resolution is a very useful 2.3mv. With such accuracy it is important to use care when connecting external devices to ensure the connections are not susceptible to interference. Interference can come from many sources including the devices own power supply which may have voltage ripples around its nominal value. As a general guide try to ensure the ground connection is particularly low impedance and perhaps use screened cable to shield the analogue signal being connected. The source of the analogue signal being measured should also have as low an output impedance as possible ensuring the analogue input does not “load it” during the sample and conversion process. Source impedances of less than 2k ohms are usually okay.



Connecting Analogue Inputs to Analogue I/O Slave

Devices which are typically connected to the analogue inputs are usually in the category of sensors. i.e they are responsible for measuring a characteristic from the “real world” and converting it into a voltage proportional to the measurement. These types of devices would measure such things as temperature, pressure, humidity, noise, light, flow rates, weight etc.. etc... The list is probably endless. However, they all have one thing in common. The voltage they produce is proportional, in some way, to the physical characteristic they are measuring. Even if the relationship is non-linear the voltage can be converted into a number and then that number can easily be scaled or otherwise arithmetically manipulated to give the desired result in the appropriate units. The job of the analogue I/O slave is to make these measured voltages available within the control program so that they can be acted on for whatever purpose that the specific application requires. The on board microcontroller takes care of the details of doing the A/D conversion leaving the programmer the simple task of just requesting a reading from the analogue inputs (see software sections).

There is one digital input on the analogue I/O slave which can be connected to a standard digital signal in the range 0 – 5v. Alternatively, with a pull-up resistor (as shown), this can also be used as a switch input.

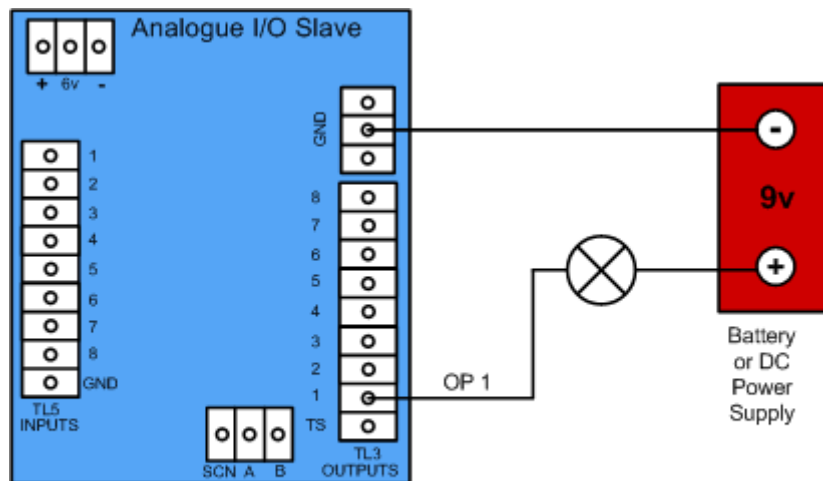


Connecting the Digital Input to Analogue I/O Slave

4.4 Outputs:

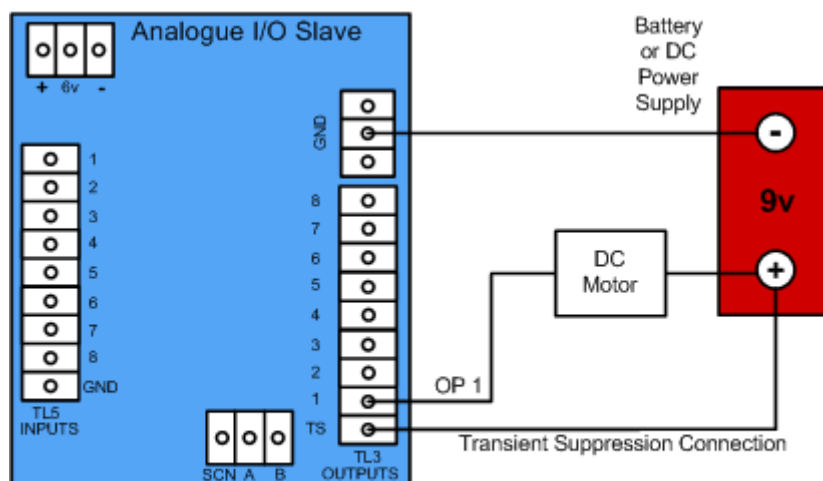
The switching outputs are slightly different to the digital inputs in their characteristics. They are “open collector” type. This means that they behave like a switch which is either open or connected to 0v. When an output is set to “On” (logic ‘1’) the switching output is closed. If you need to have a straight forward digital signal from one of these outputs, all that is necessary is to connect a resistor between the output and the positive supply of your connected device. The positive supply can be anywhere between 5v and 50v. The choice of the resistor should reflect the input characteristics of your attached device but a typical choice for a 5v system would be around 4k7.

The benefit of having open collector type outputs is that they can also be used for higher voltage and higher current switching than is typical with digital logic. For example each output on the Digital I/O Slave can switch up to 50v at 500mA. This opens the possibilities of connecting a much wider range of electrical and electronic devices directly to the outputs. These can include DC motors, solenoids, lamps, LED's, Relays etc... etc...



Connecting a Lamp to Analogue I/O Slave

By virtue of the fact you can directly drive relays from these outputs it means that devices with much higher power requirements can be driven from the controlled relay.



Connecting a DC Motor to Analogue I/O Slave

When connecting devices with an inductive load (eg motors, solenoids, relays) it is advisable to make use of the transient suppression facility provided on the board. This is

simply a series of transient suppression diodes built into the main output driver chip which act in a way as to “clamp” any transients to the positive external supply. Transient voltage spikes are a result of switching off inductive devices. The collapsing magnetic field acts in a way so as to generate much higher voltages than is normally present. These spikes can be potentially damaging to any connected components and can also cause interference to nearby RF sensitive devices such as a radio. To make use of the suppression facility, simply connect the transient suppression pin directly to the positive terminal of the external supply.

4.5 Analogue I/O Slave Connector Pinouts

Pinout of the Analogue Inputs On Screw Terminals(TL5)

Pin	Signal description
1	Analogue Input 1
2	Analogue Input 2
3	Analogue Input 3
4	Digital Input 1
5	Analogue Input 4
6	Analogue Input 5
7	Analogue Input 6
8	Analogue Input 7
GND	GND (0v)

Pinout of the High Voltage Switching Outputs On Screw Terminals(TL3)

Pin	Signal description
TS	Transient Suppression
1	Switching Output 1
2	Switching Output 2
3	Switching Output 3
4	Switching Output 4
5	Switching Output 5
6	Switching Output 6
7	Switching Output 7
8	Switching Output 8

5. Relay Slave

The Relay Slave has 8 digital inputs and 8 changeover relay outputs. Although both the Analogue I/O Slave and the Digital I/O Slave are capable of directly driving relays connected to their outputs it can be more convenient to provide this level of switching on the one board. This is the function of the Relay Slave.



5.1 General:

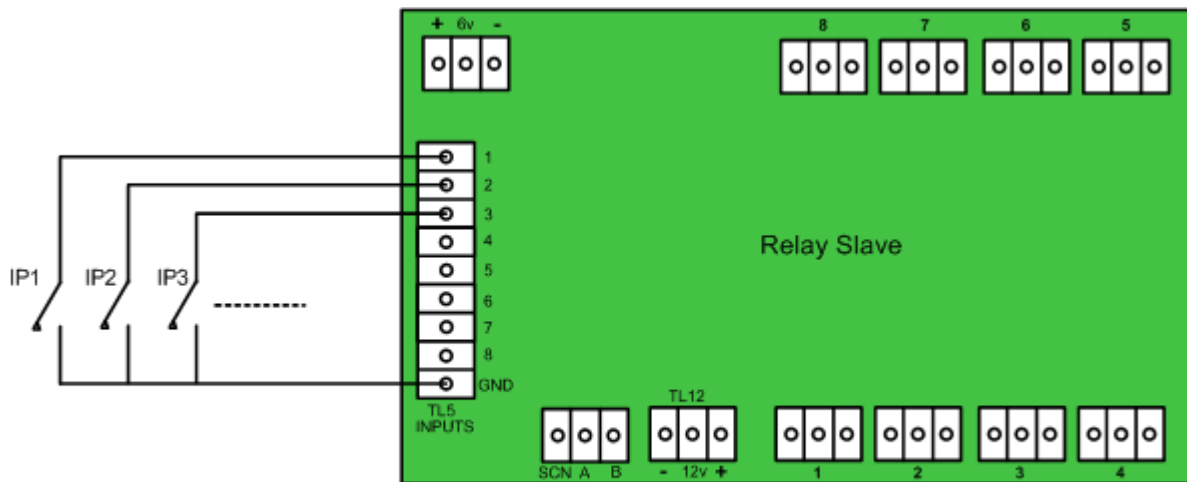
To operate correctly the Relay Slave needs to be supplied with an operating DC voltage of between 6v and 12v. This should be connected to the terminals labelled 6v (+ and -) on TL2. The power supply should be fully regulated and capable of providing at least 100mA. The Board also needs to be connected to a Master Controller using the two wires on TL1 (A and B). Connecting 'A' to 'A' and 'B' to 'B'. Alternatively it can be connected to any other slave module which is already connected to the Master using the same connection strategy. Use of the SCN connection is optional but where used it should be connected to the metal foil shielding on a twisted pair cable. . In addition to the above supply the Relay Slave needs an additional 12v DC supply to power the relay coils. This should be connected to TL12 as shown. Although it is possible to use the same 12v supply to power both the board circuitry (on TL2) and the relay coils (on TL12) it is not ideal since the switching of the relay coils on and off may cause transient voltages which may interfere with the normal operation of the board.

5.2 Board Numbering:

The board number has to be set using the blue DIL switches in exactly the same way as already described in the board numbering section of the Digital I/O Slave details.

5.3 Inputs:

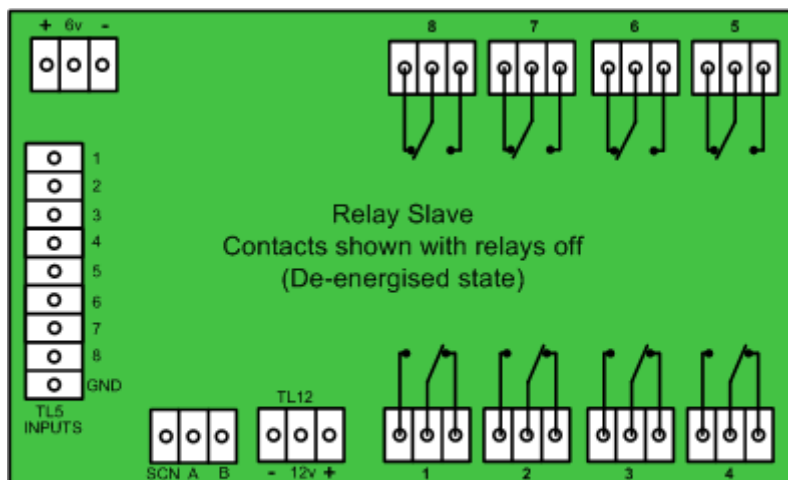
The 8 digital inputs have characteristics compatible with standard 5v logic devices. i.e. when the input is at +5v it will be read as a logic '1' or "High". When the input is at 0v or GND it will be read as logic '0' or "Low". As a convenience for use in control systems, the inputs also have "on board" pull ups. This is simply a 10K resistor connected between each of the inputs and the board +5v supply. This means that any unused inputs can be left disconnected without concern over what voltage they are "floating" to. Since the inputs are very high impedance (cmos type) they would "float" somewhere between 0 and +5v if left unconnected giving inconsistent results when read. Unconnected inputs would therefore be read as a logic '1'. The 10K resistor is sufficiently high value so as not to impose a great load on any signal source connected to it and also makes it very easy to use switches on the inputs. i.e. a switch can be connected directly between any input and the 0v (GND) connection to provide a functional digital input. When closed the input would read as logic '0', and when open as logic '1'.



Connecting Switches as Inputs to Relay Slave

5.4 Relay Outputs:

The 8 Relay outputs are available on 8 sets of 3 way screw terminals labelled simply 1 to 8. The centre terminal in each group of 3 is the changeover connection which is connected to the left hand terminal when the relay is not energised and the right terminal when energised. *Left and right are determined when looking at the wire entry face of the terminals.* Directly behind each relay is an LED indicator showing (when on) that the relay is energised. The contacts are rated for 6A at up to 250v AC (or 30v DC).



Relay Contacts Configuration on Relay Slave

Caution: Although the contacts are rated for high voltages, there is no protection provided for the exposed live terminals particularly on the underside of the board. If this board is to be used with high voltages then care should be taken to ensure all exposed terminals are made safe by using suitable enclosures or other techniques. The board should not be used "as supplied" with high voltages. If in any doubt consult a qualified electrician.

5.5 Relay Slave Connector Pinouts

Pinout of the Digital Inputs On Screw Terminals(TL5)

Pin	Signal description
1	Digital Input 1
2	Digital Input 2
3	Digital Input 3
4	Digital Input 4
5	Digital Input 5
6	Digital Input 6
7	Digital Input 7
8	Digital Input 8
GND	GND (0v)

Pinout of the Relay Contacts on Terminals 1-8
(when viewed from the cable entry side)

Left	Middle	Right
Normally Open (N/O)	Common (Com)	Normally Closed (N/C)

6. Motor Slave

The Motor Slave provides both speed and direction control for two independent DC motors with provisions for the connection of range limit switches.



6.1 General:

To operate correctly the Motor Slave needs to be supplied with an operating DC voltage of between 6v and 12v. This should be connected to the terminals labelled 6v (+ and -) on TL2. The power supply should be fully regulated and capable of providing at least 100mA. The Board also needs to be connected to a Master Controller using the two wires on TL1 (A and B). Connecting 'A' to 'A' and 'B' to 'B'. Alternatively it can be connected to any other slave module which is already connected to the Master using the same connection strategy. Use of the SCN connection is optional but where used it should be connected to the metal foil shielding on a twisted pair cable.

An additional DC supply needs to be connected to the terminals labelled VM+ and GND. This supply should be chosen according to the specifications of the motors used. It can be any voltage between 6v and 36v DC. The maximum current controlled via the Motor Slave outputs is 6A. This high current capability allows a wide range of DC motors with "useful" torque to be used for an equally wide range of applications.

Anti-static foam packing will be present between the mosfets on the board to protect these components during transit. This should be removed carefully prior to using the board.

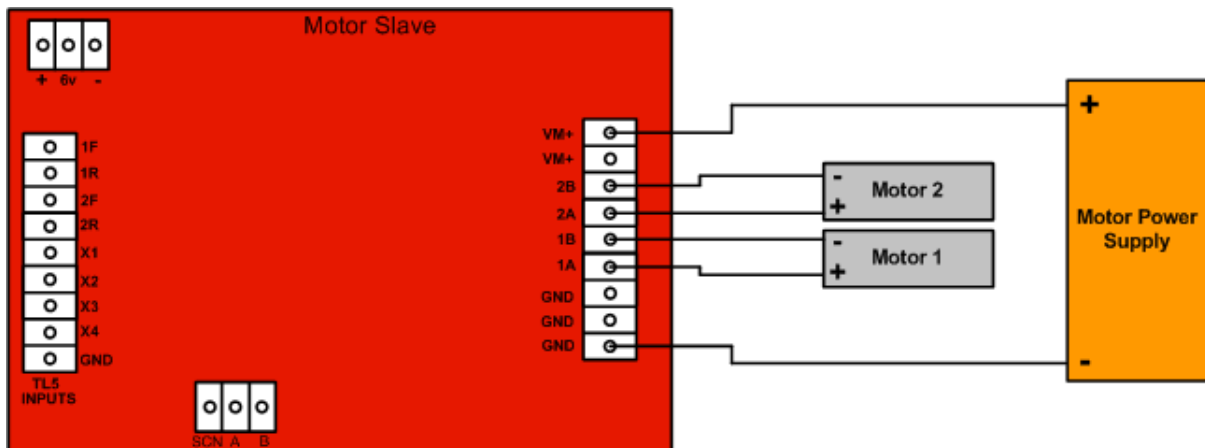


6.2 Board Numbering:

The board number has to be set using the blue DIL switches in exactly the same way as already described in the board numbering section of the Digital I/O Slave details.

6.3 Outputs:

The Motor Slave has four outputs for driving two motors independently. The output terminals are labelled as 1A and 1B for motor 1 and 2A and 2B for motor 2. When the motor is set to drive in the forward direction terminal 'A' is positive with respect to 'B' and vice-versa when set to go in reverse.



Connecting DC Motors to Motor Slave

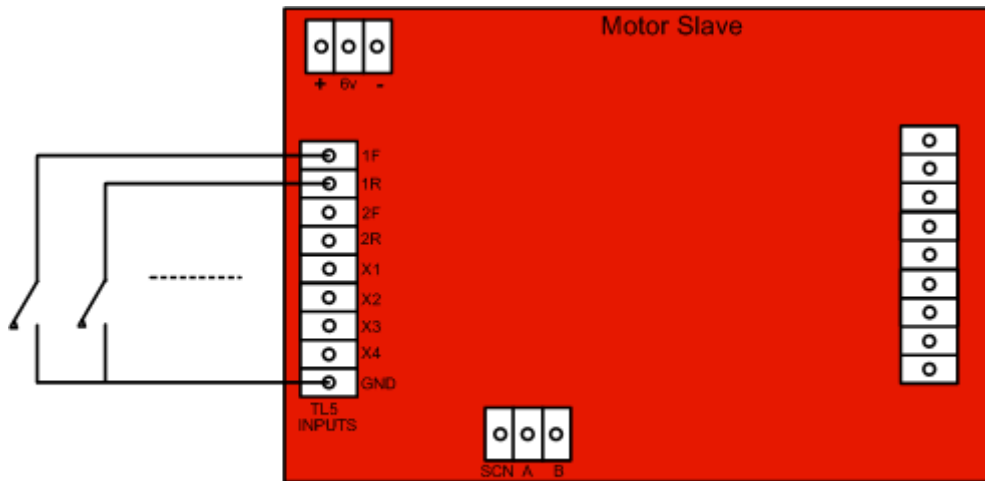
As well as direction control the motor outputs control the speed of the attached motor by using PWM (pulse width modulation) to deliver a variable power based on the external motor supply. i.e. the voltage it produces is always exactly equal to the voltage applied to the motor supply terminals (VM+ and GND) but it is constantly turned on and off at a high rate. The power transferred to the motor (and hence the resulting speed) is varied by changing the amount of time the output spends 'ON'. i.e. if the output is only on for 5ms out of every 100ms then the resulting speed would be about 5% of full speed. If it is on for 50ms out of every 100ms then you would have approx half full speed.

The speed can be varied over the full range from less than 1% to more than 99% in 255 pre-defined steps. This gives very fine control over the speed of the motor. Remember that it is the power being delivered to the motor which is being varied which in turn causes a speed change. If the motor is turning a heavy load then the speed will be proportionally less for the same power output. Each motor output is capable of delivering up to 6A to the connected motor. This allows the use of some large and powerful motors with the motor slave.

6.4 Inputs:

There are 8 digital inputs on a Motor Slave. The characteristics of these inputs are identical to those of the Digital I/O Slave already described in a previous section and will not be elaborated on here. However only 4 of these are available for general use. The four inputs labelled as X1, X2, X3 and X4 are available to be read by the controlling program at any time. The inputs labelled as 1F, 1R, 2F and 2R can also be read in the same way as X1-X4 but they also have an automatic function in the control of the motors. They are designed to act as range limits. A range limit is a mechanism to prevent a moving object moving beyond its safe operating range. For example, the motor you are controlling may be moving a drill on an X/Y drilling table along the X axis. At some point it will reach the end of its available travel and, without limits, presumably hit an end stop. If the motor continues to operate in this condition it will probably overheat and may even have enough power to damage the mechanism it is moving. To prevent this, a limit switch may be fitted near the end of travel in such a way that it is closed when reached by the moving part. The closure of this switch is used to switch off the motor automatically. However, it would be impractical to just leave the motor "dead" against the end stop with no possibility of reversing it back into the working range, so the automatic stop must only stop the forward motion. When the signal is, at some point, changed to reverse, the motor must then be allowed to reverse back from the end stop. Similarly at the other end of travel, the reverse must be inhibited automatically when the other limit switch is reached but forward motion would then be allowed. This is the function of the inputs 1F and 1R. When 1F is connected to ground it inhibits forward motion of the motor immediately and automatically without any intervention by the controlling

program. Reverse would still be allowed. Similarly 1R inhibits reverse motion but allows forward when connected to ground. 2F and 2R are the corresponding limits for motor 2. This makes the fitting of range limit switches very easy for both motors. It is not necessary to use these inputs if not required. They can simply be left unconnected for free operation in both directions since they have “pull up” resistors ensuring that they float to +5v. Since the limit switch inputs can be read like any other input it can be determined whether or not the moving device has operated any of the limit switches from within the control program allowing suitable remedial action to be taken if required.



Connecting Switches as Inputs to Motor Slave

The range limit inputs can also be used for a different purpose. They can be used as a datum point for subsequent motion. In other words the motor could deliberately be driven until the moving object reaches the limit switch which is set at a known location. This can then be used to “zero” a counter or external distance encoding device prior to subsequent movement in the opposite direction. This would give a repeatable “datum” point for more accurate position control.

6.5 Motor Slave Connector Pinouts

Pinout of the Inputs On Screw Terminals(TL5)

Pin	Signal description
1F	Motor 1 Forward Inhibit
1R	Motor 1 Reverse Inhibit
2F	Motor 2 Forward Inhibit
2R	Motor 2 Reverse Inhibit
X1	Uncommitted Digital Input X1
X2	Uncommitted Digital Input X2
X3	Uncommitted Digital Input X3
X4	Uncommitted Digital Input X4
GND	GND (0v)

Pinout of the Motor Outputs On Screw Terminals(TL3)

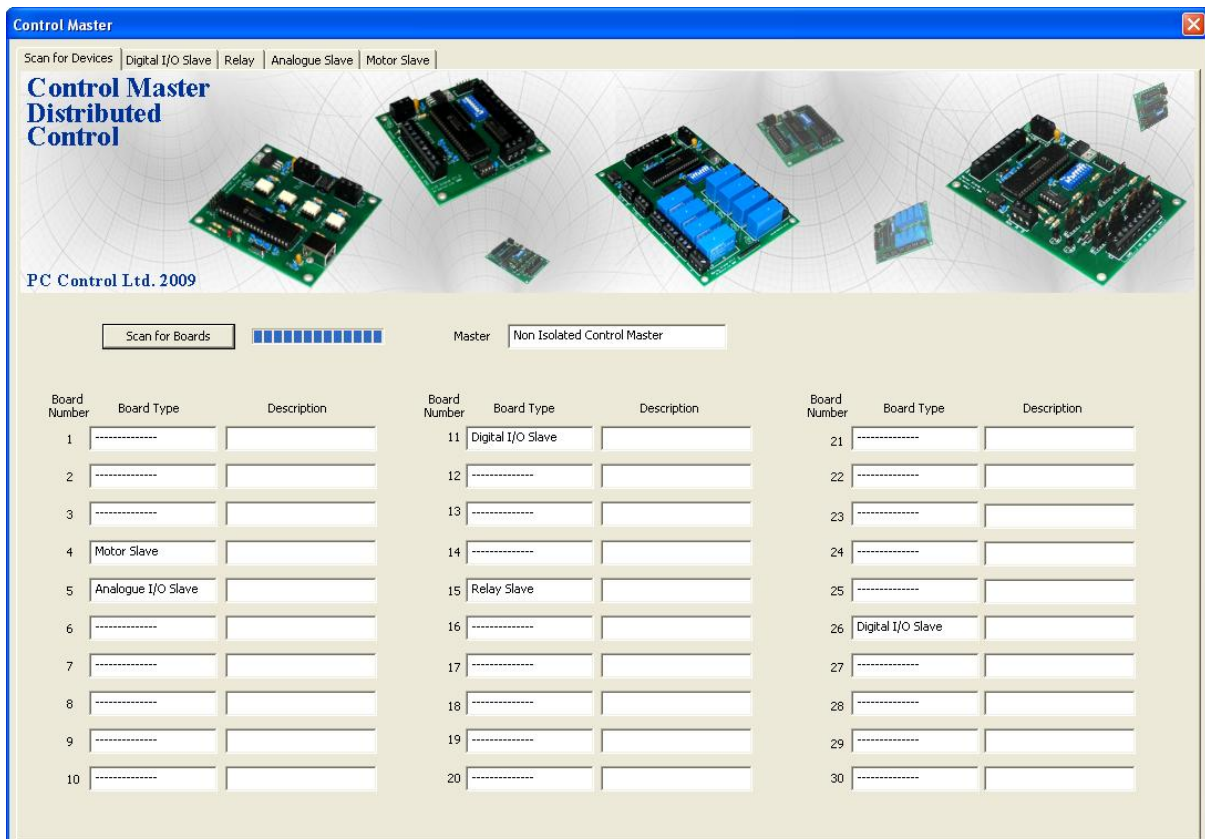
Pin	Signal description
VM+	External DC Supply For Motors (+)
VM+	External DC Supply For Motors (+)
2B	Motor 2 Connection
2A	Motor 2 Connection
1B	Motor 1 Connection
1A	Motor 1 Connection
GND	GND (0v)
GND	GND (0v)
GND	GND (0v)

7. Control Master Software

The software supplied with the Control Master range of boards consists of two parts.

The first is a “ready to run” Windows application that allows the user to access all of the facilities on all of the connected boards. It provides manual control for all of the outputs whatever type they are (simple digital, relay, motor speed and direction etc..) and a display of the current state of the inputs (digital, analogue etc...).

The second is a DLL function library to provide the programmer with an easy to use interface to all of Control Master facilities. The DLL provides functions which allow the programmer to ignore all of the complexities of USB communications and RS485 serial communications and just specify what they want a particular output to do on a specific board as well as read the current state of any input on any board.



When you first run Control Master you will be asked to agree to the terms of use of the software. Once you click “Agree” you will not see this page again. On the startup page you will see just two options “Run” and “Help”. Click on “Run” to start the main application.

Control Master is divided into separate pages which correspond to each type of slave board currently available within the Control master range. i.e. there is a separate page for the Digital I/O Slave, the Analogue I/O Slave, Motor Slave etc... As more slave board types are added to the range a new page will be added. In this way backward compatibility will be maintained with existing slave board types within Control Master.

These pages are accessed simply by clicking on the “Tab” at the top of each page. However, the first page you see when you start the application is slightly different. It is a summary page which allows you to see which Slave boards are currently connected to your system and what Board Number they have been set to. Clicking on the “Scan” button will cause the program to search through all available board number slots (1 to 30) and establish communication with any board present. It takes a few seconds and you should then see a list of the attached boards. If you do not see what you expect double check your connection details and the allocation of board numbers. The scan will also reveal what type of Master

Controller you have connected to your USB port (i.e. Isolated or Non-Isolated). If no Master is detected then, obviously, no slaves will be detected and you should check you connection of the Master to the USB port and the installation of its drivers.

Based on what board types you have connected to your system (revealed by the scan) you should then go to the page for that type of board. It should be noted that it does not matter how many of each type of board you have connected, you will still be able to access all boards connected to the system of a given type from the same page. In fact, you could have a system with 30 boards of the same type. In that case you would only use the one page for all of your boards.

7.1 Common Controls:

Each page of each board type has its own selection of controls appropriate to the specific type of board. These controls will be discussed in detail in the following sections, but, in this section we will look at the controls which are common to all board types.

Once you are on the page for your chosen board type, you need to set the Board Number of the specific board you wish to communicate with. This is done by using the up and down arrow keys adjacent to the "Board Number" display or simply by overtyping the number followed by pressing the TAB key. With the board number selected you can now start using the specific controls for that board e.g. "Update". Whenever you use a control to communicate with a board the success or otherwise of the action is displayed in a message box called "Status". If you were successful the box will display "No Errors" in green. If there was a problem it will give you some indication as to the nature of the problem and will display this in red. The type of problems it will detect and report are...

"No Board Detected with that number" – Self explanatory. You should check the scan page to make sure you have the correct board number.

"Wrong board type at specified number" – You have attempted to control one board type from a page designed for another type. e.g. using the Motor Slave page to control an Analogue I/O Slave

Since each of the Slave boards are powered entirely independently of the controlling PC it is possible for the PC to be turned off and back on again or just the Control Master application to be re-started without the attached slaves being restarted. i.e. the slaves will continue to "hold" their current output states until they are told otherwise by the controlling PC. This means that on re-starting the Control Master application the default state of the outputs on each board type will not reflect their true state. To correct this situation there is a button called "Get Settings from Board" which, when pressed, will read the current state of the selected board outputs and update the settings within Control Master. Once updated, you can then continue to use the outputs as normal. Obviously this does not apply to any inputs which would be updated as normal during the next read of the board.

Next to every input and output control on every board page there is a box for a "Description". This is intended to allow you to customise Control Master for your own particular application. They are simple text entry boxes which automatically retain the information you type into them even after a program restart. For example: You may be using a Digital I/O Slave to control two DC motors, a solenoid and three indicator lamps on the first 6 outputs. The description next to these could be...

Output	Description
1	Motor 1 (X axis)
2	Motor 2 (Y axis)
3	Pick up solenoid
4	Motion complete indicator lamp
5	Traversing error indicator

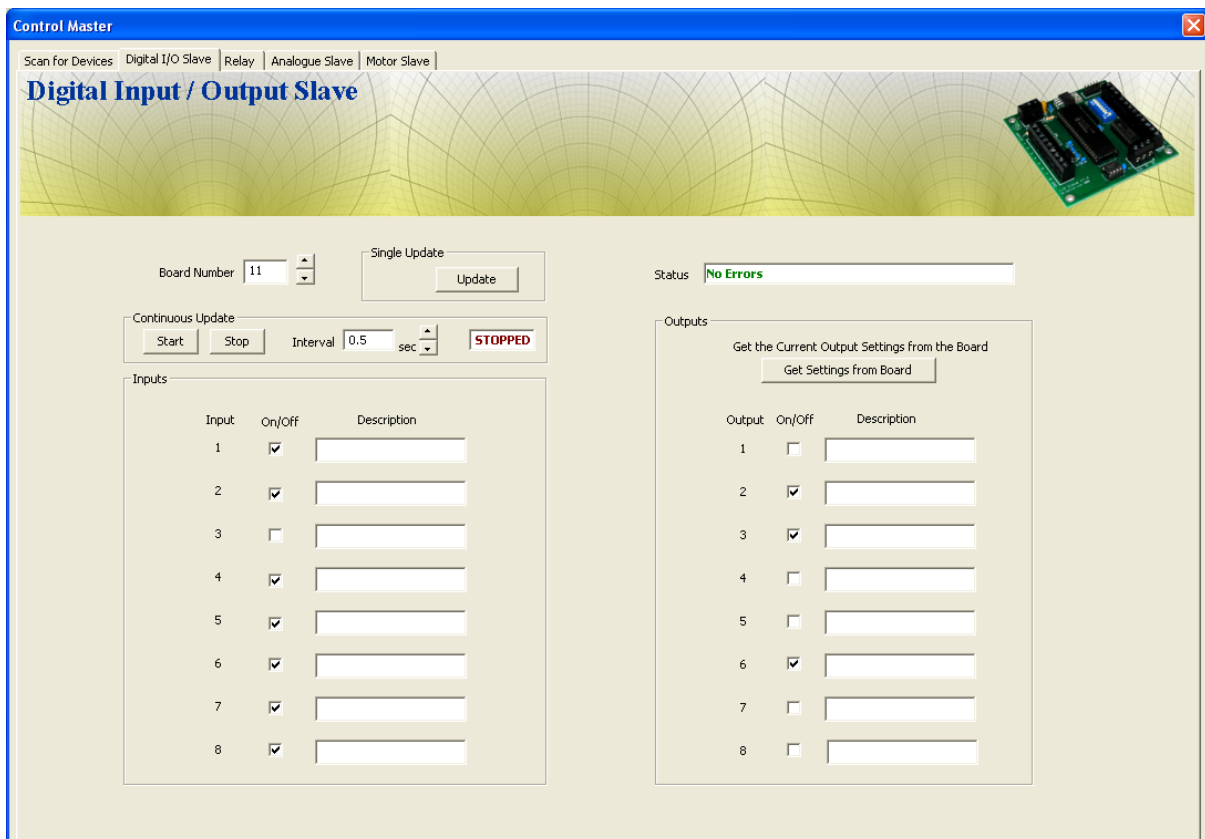
This makes your control system much more accessible to understanding.

Since every board of the same type is accessed by the same page on Control Master it is necessary to maintain a separate set of descriptions for each board of the same type on that boards page, since it is quite possible, and likely, that the same output or input on a different board is used for a different purpose. This is supported automatically by changing the descriptions every time you change the board number. No data you enter into the description boxes is ever lost or ever has to be manually saved. This is done fully automatically within the program.

It is also possible to add descriptive text to the list of boards found on the scan page. Here you can enter descriptions meaningful to you about each board. For example you could add a description showing the location of each board. "Workshop 1", "Garage", "Greenhouse", "Camera1 Tilt and Swivel" etc... etc... Again these descriptions, once entered, are automatically saved when you exit the program and automatically restored on restart.

7.2 Digital Input / Output Page

Controls specific to the Digital I/O Page are detailed in this section. If the control you are looking for is not mentioned then it will most likely be covered in the previous section on controls common to all pages.



The Digital I/O Slave has 8 inputs and 8 outputs and control of these from its control page is very simple indeed. Before using any of the controls it is necessary to set the Board Number to the specific board you want to control. This can be set by simply overtyping the board number shown followed by "TAB" or by using the up/down arrow keys to the right of the number. You can always check that you have the correct number by viewing the results of a scan on the scan page. Within the Group box labelled outputs are 8 tick boxes

corresponding to outputs 1 to 8 respectively. To turn an output on, simply tick the relevant box and press the “Update” button. Ticking the box again will turn the output off the next time you press “Update”.

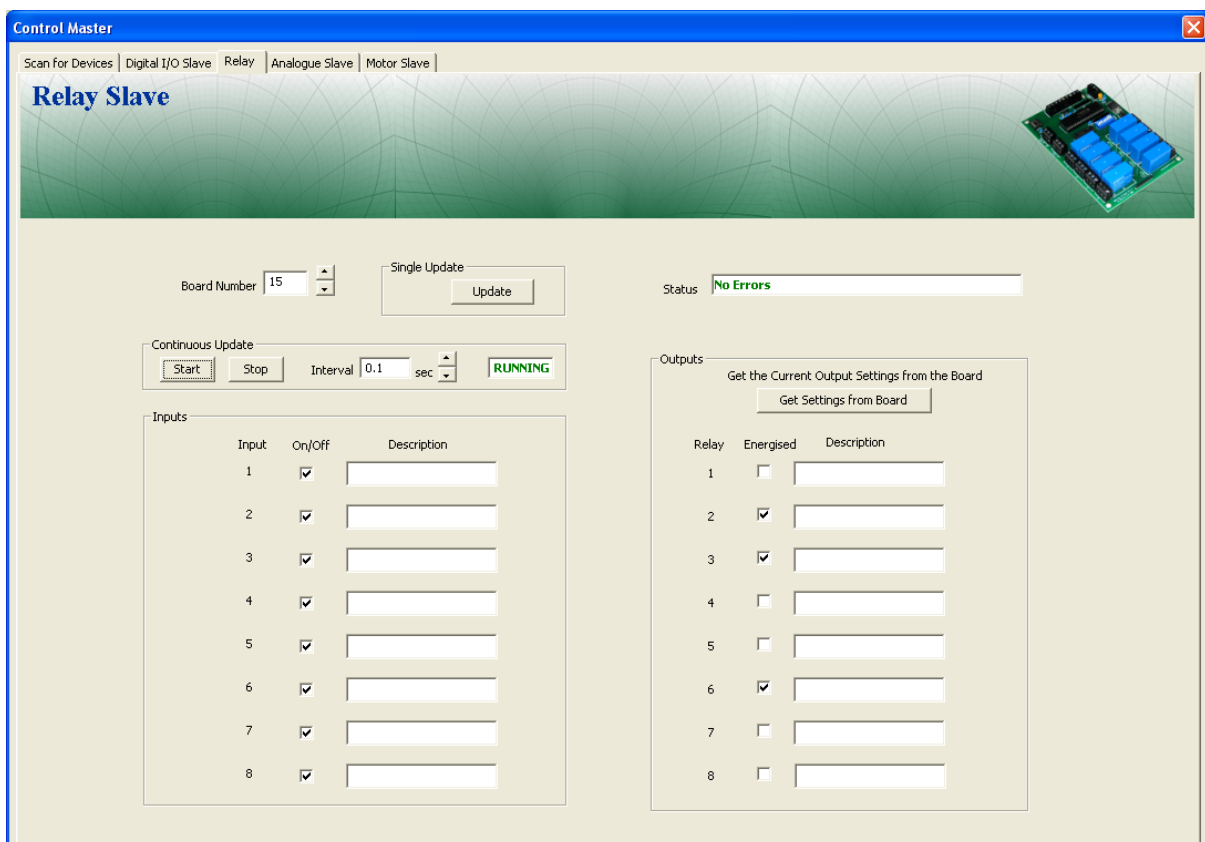
At the same time outputs are set by pressing the “Update” button, the inputs are also read and updated to the screen. These are shown in the group box labelled inputs. In the same way that an output is represented by a tick in a box, so are the inputs. A tick represents a logic ‘1’, “High”, “+5v” on the corresponding input. No tick means the input is at logic ‘0’, “Low”, 0v. If you only need to read the inputs and do not want to change any outputs, then simply press the “Update” button. As long as the outputs still have the same tick marks present then no output will be changed. i.e. even though the outputs get set every time you click “Update” there will be no problems in repeating this several times as you monitor the inputs.

As a convenience you can choose to continually “Update” at regular intervals by using the “Continuous Update” option. Simply set the interval you prefer and click “Start” to have regular “Updates” automatically. Even while continuously updating you can still tick and untick the output boxes and see the changes take effect during the next timed update.

The automatic interval timer may be stopped at any time by pressing the “Stop” button. By changing to a different page, the continuous updates will be stopped automatically.

7.3 Relay Page

Controls specific to the Relay Page are detailed in this section. If the control you are looking for is not mentioned then it will most likely be covered in the previous section on controls common to all pages.



The Relay Slave has 8 inputs and 8 relay outputs and control of these from its control page is very simple indeed. Before using any of the controls it is necessary to set the Board

Number to the specific board you want to control. This can be set by simply overtyping the board number shown followed by “TAB” or by using the up/down arrow keys to the right of the number. You can always check that you have the correct number by viewing the results of a scan on the scan page. Within the Group box labelled relay outputs are 8 tick boxes corresponding to relay outputs 1 to 8 respectively. To turn a relay output on, simply tick the relevant box and press the “Update” button. Ticking the box again will turn the output off the next time you press “Update”.

At the same time relay outputs are set by pressing the “Update” button, the inputs are also read and updated to the screen. These are shown in the group box labelled inputs. In the same way an output is represented by a tick in a box, so are the inputs. A tick represents a logic ‘1’, “High”, “+5v” on the corresponding input. No tick means the input is at logic ‘0’, “Low”, 0v. If you only need to read the inputs and do not want to change any outputs, then simply press the “Update” button. As long as the outputs still have the same tick marks present then no output will be changed. i.e. even though the outputs get set every time you click “Update” there will be no problems in repeating this several times as you monitor the inputs.

As a convenience you can choose to continually “Update” at regular intervals by using the “Continuous Update” option. Simply set the interval you prefer and click “Start” to have regular “Updates” automatically. Even while continuously updating you can still tick and untick the output boxes and see the changes take effect during the next timed update.

The automatic interval timer may be stopped at any time by pressing the “Stop” button. By changing to a different page, the continuous updates will be stopped automatically.

7.4 Analogue Input / Output Page

Controls specific to the Analogue I/O Page are detailed in this section. If the control you are looking for is not mentioned then it will most likely be covered in the previous section on controls common to all pages.

The screenshot displays the 'Analogue Input / Output Slave' control page in the 'Control Master' software. At the top, navigation tabs include 'Scan for Devices', 'Digital I/O Slave', 'Relay', 'Analogue Slave', and 'Motor Slave'. The main title is 'Analogue Input / Output Slave' with a small image of a circuit board. Below the title, the 'Board Number' is set to 5. There are two update modes: 'Single Update' with an 'Update' button, and 'Continuous Update' with 'Start' and 'Stop' buttons, an interval of 0.2 seconds, and a 'STOPPED' status indicator. The 'Status' field shows 'No Errors'. The main area contains seven vertical sliders for analogue inputs, each with a scale from 0 to 1000 and a corresponding H.S. value: 714, 660, 374, 273, 579, 504, and 335. Each slider also has 'Offset' and 'Scale' fields, all set to 0 and 1 respectively. Below the sliders are 'Meas' and 'Units' fields for each input. On the right, there is a 'Digital Input' field set to 1 and a table of eight digital outputs with 'On/Off' checkboxes. A 'Get Settings from Board' button is located at the bottom right.

The Analogue I/O Slave has 7 analogue inputs, 1 digital input and 8 switching outputs. Before using any of the controls it is necessary to set the Board Number to the specific board you want to control. This can be set by simply overtyping the board number shown followed by "TAB" or by using the up/down arrow keys to the right of the number. You can always check that you have the correct number by viewing the results of a scan on the scan page.

Within the Group box labelled outputs are 8 tick boxes corresponding to outputs 1 to 8 respectively. To turn an output on, simply tick the relevant box and press the "Update" button. Ticking the box again will turn the output off the next time you press "Update".

At the same time outputs are set by pressing the "Update" button, the analogue inputs are also read and updated to the screen.

The analogue inputs are represented in a number of ways on the display screen. Each analogue input has a vertical slider representation. This shows a moving height pointer indicating the current measured input as a change in height. The bottom of the slider corresponds to 0v and the top to +5v (normally) with a range of 1023 steps in between. The 5v maximum can be replaced with a higher sensitivity option for each analogue input by ticking the H.S. box at the bottom of the slider. This makes the full height reading correspond to 2.4v (rather than 5v) with the bottom remaining 0v. In this mode there are still 1023 steps between limits giving the measurement a very fine resolution of 2.3mv. As well as the height changing pointer there is also a numeric indication of the measured voltage shown at the bottom of the slider. This figure corresponds to the "raw" A/D convertor output. i.e. it will be a number in the range 1 – 1023 since it is a 10 bit convertor.

When a measurement is being made of a voltage appearing at one of the analogue inputs it is usually related to some physical quantity such as temperature, humidity, weight etc.. resulting from the output of a sensor. When this is the case, it can be more useful to see the measured value on the screen in the same units as the physical quantity being measured. For this reason four additional boxes are provided. The first two "Offset" and "Scale" should contain details of the calculation involved in converting from the "raw" A/D number to the correct physical unit. The third box "Meas" will automatically display the result of this calculation and the fourth "Unit" is simply a place to type the abbreviation for the units used (eg mm, deg etc). The way the first two boxes operate is to, first of all, multiply the "raw" A/D result by the figure in the "Scale" box and then add the number in the "Offset" box. The resulting answer is then shown in the "Meas" box. Choosing the correct figures is a task you need to do based on the characteristics of your sensor and the chosen voltage range for the A/D (i.e. 2.4v or 5v). To illustrate this technique lets look at a simple example.

Lets say you have a temperature sensor that produces an output of 0 to 5v over the temperature range 10 degC to 80 degC. Therefore when the voltage is 0v (and the A/D output is 0) the temperature should read "10". Your Offset value should therefore be chosen to be "10". When the voltage is 5v the temperature should read 80, but the A/D output is 1023. The value 1023 corresponds to a change in temperature of 70 deg, therefore....

$$\begin{aligned} \text{Scale} &= 70 / 1023 \\ \text{Scale} &= 0.068 \end{aligned}$$

Your "Scale" value should be chosen to be 0.068. Once these figures have been entered in the two boxes, changes in the temperature will now automatically be displayed correctly in degrees in the "Meas" display box.

It should be noted that it may be necessary, depending on your sensor, to subtract an offset. In this case simply enter a negative number in the offset box, i.e. precede it with a '-'

The one digital input is shown directly above the outputs section. A '1' represents a logic '1', "High", "+5v" and a '0' means the input is at logic '0', "Low", 0v.

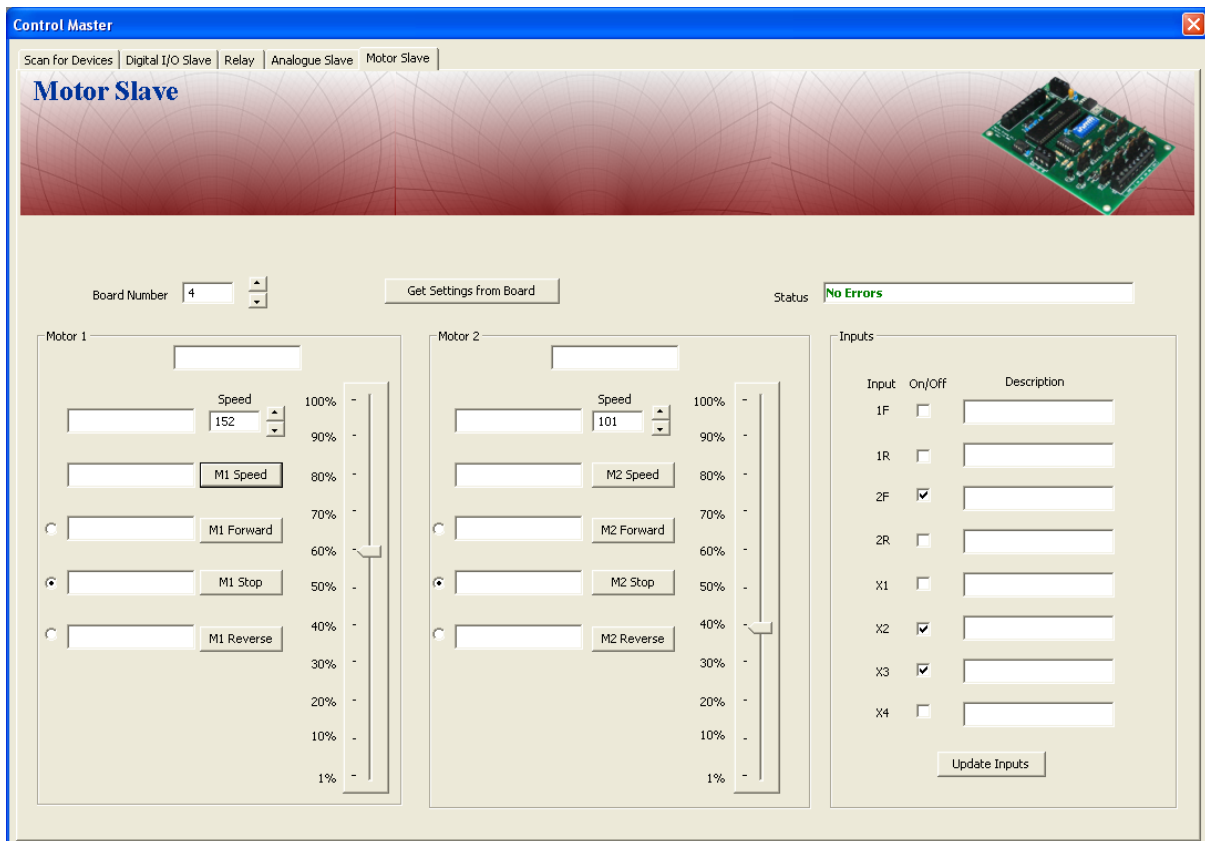
If you only need to read the inputs and do not want to change any outputs, then simply press the “Update” button. As long as the outputs still have the same tick marks present then no output will be changed. i.e. even though the outputs get set every time you click “Update” there will be no problems in repeating this several times as you monitor the inputs.

As a convenience you can choose to continually “Update” at regular intervals by using the “Continuous Update” option. Simply set the interval you prefer and click “Start” to have regular “Updates” automatically. Even while continuously updating you can still tick and untick the output boxes and see the changes take effect during the next timed update.

The automatic interval timer may be stopped at any time by pressing the “Stop” button. By changing to a different page, the continuous updates will be stopped automatically.

7.5 Motor Page

Controls specific to the Motor Page are detailed in this section. If the control you are looking for is not mentioned then it will most likely be covered in the previous section on controls common to all pages.



The Motor Slave has 8 inputs, 4 of which have special functions local to the board, and 2 Motor Control outputs. Before using any of the controls it is necessary to set the Board Number to the specific board you want to control. This can be set by simply overtyping the board number shown followed by “TAB” or by using the up/down arrow keys to the right of the number. You can always check that you have the correct number by viewing the results of a scan on the scan page.

The controls for the Motors connected to the Motor Slave are very simple. On the Motor Page screen they are divided into three areas: The inputs, the controls for Motor 1 and

the controls for Motor 2. For the purposes of brevity the following descriptions will use motor 1 controls as the example, but the descriptions apply equally to motor 2 controls.

Control consists of two parts "Speed" and "Direction". Speed is chosen by either moving the vertical slider, overtyping the speed figure in the box or by using the arrow keys adjacent to the speed figure. Once the desired speed figure is displayed simply press the "M1 Speed" button to command the motor to run at that speed. Note that if the motor is currently stationary it will have no effect. The direction would then need to be chosen by pressing "M1 Forward" or "M1 Reverse". The figure used for representing speed is always in the range of 1 to 255. The absolute value of this number does not have any significance other than to say that it represents the figure used for generating the PWM control on the Motor Slave board. A "percentage of full speed" equivalent of this figure is what is shown on the vertical slider.

Once the motor is running you can stop it at any time by pressing "M1 Stop". Similarly the direction of each motor can be controlled by pressing the buttons marked "M1 Forward" and "M1 Reverse". Pressing these buttons has an immediate effect on the motor. When choosing direction, no change is made to the speed of the motor.

Although it is perfectly possible to change from full speed forward to full speed reverse simply by pressing the "M1 Reverse" button, you may find that this will put substantial stresses on your motor (especially those with a higher power rating or with a higher inertia due to any attached loads etc.). A better strategy is to first stop the motor *using "M1 Stop" and then press the desired direction button (e.g. "M1 Reverse").

In the description of the Motor Slave hardware in a previous section you will have found out about the range limit inputs which operate automatically to stop the motor when necessary. These inputs form the first four as shown in the inputs section. The second four inputs labelled X1 to X4 are general purpose and available for any use. To update these inputs, simply press the "Update" button. A tick represents a logic '1', "High", "+5v" on the corresponding input. No tick means the input is at logic '0', "Low", "0v". The "Update" button may be pressed at any time. However, due to the nature of the speed control facilities on the Motor Slave (PWM) you may notice a slight hesitation in the constant speed of a motor as the read inputs request is processed by the board. This is usually imperceptible. It only becomes slightly noticeable when using very sensitive motors with almost no load or when running a motor at very low speeds and the read inputs request is sent very frequently.

8. Programmers Guide

Whatever programming language you use it is necessary to ensure your compiler is configured to use the `__stdcall` calling convention (rather than `__fastcall` or `__cdecl`) when calling functions in an external DLL. This option is usually found within the Code Generation section of your programming environment. For example, with Microsoft Visual C++ Studio it is found in ...

Project Properties / C++ / Advanced / Calling Convention

It is also necessary to check that your compiler is using 4 byte Integers, although, this is usually the default with most compilers.

To make use of the DLL function library in your own program see your programming manual on "How to Access External DLL's" or a similarly titled section. The specifics will be contained there. In general however, you will need to load the DLL into your program memory and then obtain pointers to the functions within it before you then actually start using those functions via the pointers obtained. To make the DLL accessible you need to copy it from the installation disk (the file called "mas.dll" found in the "DLL" directory) to your main hard drive. It should be copied to the same directory where you will be running your own program or, better still, to the "C:\Windows\System32" directory where it will be found by your program wherever you run it.

In C / C++ the following code would be used....

```
// a variable to hold a handle to the dll
HINSTANCE  HMasDll;

// Load the DLL and save the handle in the variable already created
HMasDll = LoadLibrary("mas.dll");

// create a pointer type definition for each specific function
// in the DLL. Note: these typedefs are supplied in a header file,
// called "masdll.h", for use with the DLL
typedef int(*Type_CM_DIO_Update)(int BoardNumber,int Outputs,int *Inputs);

// create a variable of the correct pointer type for each specific function // in the DLL.
Type_CM_DIO_Update  CM_DIO_Update;

// initialize that pointer variable with the address of the function within
// the dll CM_DIO_Update=(Type_CM_DIO_Update)GetProcAddress(HMasDll,"CM_DIO_Update");

// then you are free to use the function anywhere within your program
// for example

int BoardNumber;
int Outputs;
int Inputs;

Outputs = 0x55;          // alternate on and off across the 8 outputs

// Update the board (i.e. send the new outputs and read back the current inputs
CM_DIO_Update=(BoardNumber, Outputs, &Inputs);

// After the above call, the variable inputs would hold the current pattern
//of inputs appearing at the board
```

In Basic / Visual Basic the much simpler code would be used....

```
Declare Function CM_DIO_Update Lib "mas.dll" (ByVal BoardNumber As Integer, ByVal Outputs As Integer, ByRef Inputs As Integer) As Integer
```

The DLL mas.dll should be available within the working directory of your program or, better still, in your "C: \ Windows \ System32" directory , where it will be found wherever you run your program.

The function CM_DIO_Update is now ready for use within the program in a similar way to the 'C' version shown above. i.e.

```
Dim BoardNumber As Integer
Dim Inputs As Integer
Dim Outputs As Integer
```

```
BoardNumber = 15
Outputs = 85          ' outputs 1,3,5 and 7 On, 2,4,6 and 8 off
```

```
' Update the board (i.e. send the new outputs and read back the current inputs
CM_DIO_Update( BoardNumber, Outputs, Inputs )
```

```
' After the above call, the variable inputs would hold the current pattern of inputs appearing at the board
```

8.1 DLL Functions Reference

This section will explain the details of each function provided within the DLL library and how to use them from within your own program. The functions within the DLL have names which reflect their usage. All of the names start with CM_ indicating a “Control Master” function. The next group of letters indicates the type of slave board that it applies to. For example CM_RELAY_ is , quite obviously used with a relay slave board. The remaining text describes the function itself. E.g. CM_RELAY_Update(.....). There are a few functions which are not for specific slave boards and these do not have the second group of letters after the CM. e.g. CM_Initialise(), CM_GetBoardType(.....).

Starting with the functions not specific to any slave board, the following is a reference list of the DLL functions grouped by applicable board type.

Function Name:	CM_Initialise
Applicable to:	General System
Syntax ‘C / C++’:	int CM_Initialise()
Syntax ‘Basic/VB’:	CM_Initialise () As Integer
Return Value:	‘0’ if successful and a negative number if an error occurs

Description: This function must be called just once per program instance. It must be called prior to any other DLL function being used. Its function is to establish communications links with the Master Controller across the USB link and prepare the system to accept all other function calls. A list of possible errors can be found in the dll header file “masdll.h”

Usage “C / C++”:

```
int error;
```

```
error = CM_Initialise();
if(error == 0)
{
    ..... Main Program
}
else
{
    ..... Report error and stop
}
```

Usage “Basic / Visual Basic”:

```
Dim error As Integer
```

```
error = CM_Initialise()

If error = 0 Then
    ..... Main Program
Else
    ..... Report error
End If
```

Function Name: CM_GetBoardType
Applicable to: General System
Syntax 'C / C++': int CM_GetBoardType(int BoardNumber)
Syntax 'Basic/VB': CM_GetBoardType (ByVal BoardNumber As Integer) As Integer

Return Value: returns the type number of the board detected at the number specified. This will be a number in the range 0 – 99.
returns the number “123” if no board is detected at that number

Description: This function can be called to determine what board is present (if any) at the specified number. It will return the board type number if successful. If no board is present at the specified board number or the system cannot establish reliable communications with it, it will return the number “123”. This number is outside the valid range of board types (0-99). In the special case of board number '0' it will return the type of Master connected to the USB port. i.e. Isolated or Non-Isolated. A list of all relevant board types can be found in the dll header file “masdll.h”.

Usage “C / C++”:

int BoardType, BoardNumber;

BoardNumber = 15; // find the type of board connected whose board number switch is set to15

BoardType = CM_GetBoardType(BoardNumber);

Usage “Basic / Visual Basic”:

Dim BoardNumber As Integer

Dim BoardType As Integer

BoardNumber = 15 ' find the type of board connected whose board number switch is set to15

BoardType = CM_GetBoardType(BoardNumber)

Function Name: CM_DIO_Update
Applicable to: Digital Input / Output Slave
Syntax 'C/C++': int CM_DIO_Update (int BoardNumber, int Outputs, int *Inputs)
Syntax 'Basic/VB': CM_DIO_Update (ByVal BoardNumber As Integer,
ByVal Outputs As Integer,
ByRef Inputs As Integer) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will carry out an update on the digital I/O slave specified by Board Number. The outputs specified will be applied to the physical boards outputs and the current board inputs will be returned in the "inputs" variable. Bits 0 to 7 of Inputs correspond to inputs 1 to 8 respectively with a '1' being "High" (i.e. "+5v").

Usage "C / C++":

int Inputs, Outputs, Error;

BoardNumber = 5;

Outputs = 0x0F; // outputs 1,2,3 and 4 On, 5,6,7 and 8 off

Error = CM_DIO_Update(BoardNumber, Outputs, &Inputs);

If(Error == 0)

{

Use Inputs // Inputs now contains valid data read from board 5

}

Usage "Basic / Visual Basic":

Dim BoardNumber As Integer

Dim Inputs As Integer

Dim Outputs As Integer

BoardNumber = 15

Outputs = 85 ' outputs 1,3,5 and 7 On, 2,4,6 and 8 off

Error = CM DIO_Update(BoardNumber, Outputs, Inputs)

If Error = 0 Then

Use Inputs ' Inputs now contains valid data read from board 15

End If

Function Name: CM_DIO_GetStatus

Applicable to: Digital Input / Output Slave

Syntax 'C/C++': int CM_DIO_GetStatus (int BoardNumber, int *Outputs)

Syntax 'Basic/VB': CM_DIO_GetStatus (ByVal BoardNumber As Integer,
ByRef Outputs As Integer) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will do the opposite of an update on the digital I/O slave. It will read back the current state of the Outputs on the board specified by Board Number. This is designed for the situation where the PC has been turned off or the application program has been terminated but the slave board remains on. It allows the PC to update its record of the outputs before resuming changes to them. Bits 0 to 7 of Outputs correspond to outputs 1 to 8 respectively with a '1' being "on".

Usage "C / C++":

```
int Outputs, Error;
```

```
BoardNumber = 25;
```

```
Error = CM_DIO_GetStatus( BoardNumber, &Outputs);
```

```
If(Error == 0)
```

```
{
```

```
    Use Outputs           // Outputs now contains valid outputs as read from board 25
```

```
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
```

```
Dim Outputs As Integer
```

```
BoardNumber = 25
```

```
Error = CM DIO_GetStatus( BoardNumber, Outputs)
```

```
If Error = 0 Then
```

```
    Use Outputs           ' Outputs now contains valid outputs as read from board 25
```

```
End If
```

Function Name: CM_RELAY_Update
Applicable to: Relay Slave
Syntax 'C/C++': int CM_RELAY_Update (int BoardNumber, int Outputs, int *Inputs)
Syntax 'Basic/VB': CM_RELAY_Update (ByVal BoardNumber As Integer,
ByVal Outputs As Integer,
ByRef Inputs As Integer) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will carry out an update on the relay slave specified by Board Number. The outputs specified will be applied to the physical boards outputs and the current board inputs will be returned in the "inputs" variable. Bits 0 to 7 of Inputs correspond to inputs 1 to 8 respectively with a '1' being "High" (i.e. "+5V").

Usage "C / C++":

int Inputs, Outputs, Error;

BoardNumber = 5;

Outputs = 0x0F; // outputs 1,2,3 and 4 On, 5,6,7 and 8 off

Error = CM_RELAY_Update(BoardNumber, Outputs, &Inputs);

If(Error == 0)

{

Use Inputs // Inputs now contains valid data read from board 5

}

Usage "Basic / Visual Basic":

Dim BoardNumber As Integer

Dim Inputs As Integer

Dim Outputs As Integer

BoardNumber = 15

Outputs = 85 // outputs 1,3,5 and 7 On, 2,4,6 and 8 off

Error = CM_RELAY_Update(BoardNumber, Outputs, Inputs)

If Error = 0 Then

Use Inputs ' Inputs now contains valid data read from board 15

End If

Function Name: CM_RELAY_GetStatus

Applicable to: Relay Slave

Syntax 'C/C++': int CM_RELAY_GetStatus (int BoardNumber, int *Outputs)

Syntax 'Basic/VB': CM_RELAY_GetStatus (ByVal BoardNumber As Integer,
ByRef Outputs As Integer) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will do the opposite of an update on the relay slave. It will read back the current state of the Outputs on the board specified by Board Number. This is designed for the situation where the PC has been turned off or the application program has been terminated but the slave board remains on. It allows the PC to update its record of the outputs before resuming changes to them. Bits 0 to 7 of Outputs correspond to outputs 1 to 8 respectively with a '1' being "on".

Usage "C / C++":

```
int Outputs, Error;
```

```
BoardNumber = 17;
```

```
Error = CM_RELAY_GetStatus( BoardNumber, &Outputs);
```

```
If(Error == 0)
```

```
{
```

```
    Use Outputs           // Outputs now contains valid outputs as read from board 17
```

```
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
```

```
Dim Outputs As Integer
```

```
BoardNumber = 2
```

```
Error = CM_RELAY_GetStatus( BoardNumber, Outputs)
```

```
If Error = 0 Then
```

```
    Use Outputs           ' Outputs now contains valid outputs as read from board 2
```

```
End If
```

Function Name: CM_ANIO_Update1to4
Applicable to: Analogue Input / Output Slave
Syntax 'C/C++': int CM_ANIO_Update1to4 (int BoardNumber, int Outputs, int *Inputs, int Sensitivity)

Syntax 'Basic/VB': CM_ANIO_Update1to4 (ByVal BoardNumber As Integer, ByVal Outputs As Integer, ByRef Inputs As Integer, ByVal Sensitivity As Integer) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will carry out an update on an analogue I/O slave specified by Board Number. The outputs specified will be applied to the physical boards outputs and the digitally converted values of the first 4 analogue inputs will be returned in the "inputs" variable. The "inputs" pointer supplied should point to an array of 4 integers since this function will return 4 integers starting at the pointer address. The first 4 bits of the Sensitivity variable are used to specify the sensitivity setting on each of the four analogue inputs. A bit value of '1' indicates "High Sensitivity" and a "0" Low Sensitivity. Bit 0 corresponds to analogue input 1, Bit 1 to analogue input 2 etc....

Usage "C / C++":

```
int Inputs[4], Outputs, Sensitivity, Error;
```

```
BoardNumber = 5;  
Outputs = 0x0F; // outputs 1,2,3 and 4 On, 5,6,7 and 8 off  
Sensitivity = 0x0003; // analogue inputs 1 and 2 are high sensitivity (3 & 4 are normal)
```

```
Error = CM_ANIO_Update1to4( BoardNumber, Outputs, Inputs, Sensitivity );
```

```
If(Error == 0)  
{  
    Use Inputs[4] // Inputs[] array now contains valid data read from board 5  
                // Input[0] = Analogue input 1,  
                // Input[1] = Analogue input 2,  
                // Input[2] = Analogue input 3,  
                // Input[3] = Analogue input 4,  
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer  
Dim Inputs(4) As Integer  
Dim Sensitivity As Integer  
Dim Outputs As Integer
```

```
BoardNumber = 15  
Outputs = 85 ' outputs 1,3,5 and 7 On, 2,4,6 and 8 off  
Sensitivity = 3 ' analogue inputs 1 and 2 are high sensitivity (3 & 4 are normal)
```

```
Error = CM ANIO_Update1to4( BoardNumber, Outputs, Inputs, Sensitivity )
```

```
If Error = 0 Then  
    Use Inputs() ' Inputs() array now contains valid data read from board 15
```

```
' Input(0) = Analogue input 1,  
' Input(1) = Analogue input 2,  
' Input(2) = Analogue input 3,  
' Input(3) = Analogue input 4,
```

End If

Function Name: CM_ANIO_Update5to8
Applicable to: Analogue Input / Output Slave
Syntax 'C/C++': int CM_ANIO_Update5to8 (int BoardNumber, int Outputs, int *Inputs, int Sensitivity)

Syntax 'Basic/VB': CM_ANIO_Update5to8(ByVal BoardNumber As Integer, ByVal Outputs As Integer, ByRef Inputs As Integer, ByVal Sensitivity As Integer) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will carry out an update on an analogue I/O slave specified by Board Number. The outputs specified will be applied to the physical boards outputs and the digitally converted values of the next 3 analogue inputs and the one digital input will be returned in the "inputs" variable. The "inputs" pointer supplied should point to an array of 4 integers since this function will return 4 integers starting at the pointer address. Note that the first three integers will be in the range 1 – 1023 whilst the fourth (digital input) will be either a 0 or 1 depending on the input being "high" or "low" respectively.

The first 4 bits of the Sensitivity variable are used to specify the sensitivity setting on each of the four analogue inputs. A bit value of '1' indicates "High Sensitivity" and a "0" Low Sensitivity. Bit 0 corresponds to analogue input 1, Bit 1 to analogue input 2 etc....

Usage "C / C++":

```
int Inputs[3], Outputs, Sensitivity, Error;
```

```
BoardNumber = 5;  
Outputs = 0x0F; // outputs 1,2,3 and 4 On, 5,6,7 and 8 off  
Sensitivity = 0x0005; // analogue inputs 5 and 7 are high sensitivity (6 & 8 are normal)
```

```
Error = CM_ANIO_Update5to7( BoardNumber, Outputs, Inputs, Sensitivity );
```

```
If(Error == 0)  
{  
    Use Inputs[] // Inputs[] array now contains valid data read from board 5  
                // Input[0] = Analogue input 5,  
                // Input[1] = Analogue input 6,  
                // Input[2] = Analogue input 7,  
                // Input[3] = Digital input,  
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer  
Dim Inputs(3) As Integer  
Dim Sensitivity As Integer  
Dim Outputs As Integer
```

```
BoardNumber = 15  
Outputs = 85 ' outputs 1,3,5 and 7 On, 2,4,6 and 8 off  
Sensitivity = 5 ' analogue inputs 5 and 7 are high sensitivity (6 & 8 are normal)
```

```
Error = CM ANIO_Update5to8( BoardNumber, Outputs, Inputs, Sensitivity )
```

```
If Error = 0 Then  
    Use Inputs() ' Inputs() array now contains valid data read from board 15
```

```
' Input(0) = Analogue input 5,  
' Input(1) = Analogue input 6,  
' Input(2) = Analogue input 7,  
' Input(3) = Digital input
```

End If

Function Name: CM_ANIO_GetStatus
Applicable to: Analogue Input / Output Slave
Syntax 'C/C++': int CM_ANIO_GetStatus (int BoardNumber, int *Outputs, int *Sens)
Syntax 'Basic/VB': CM_ANIO_GetStatus (ByVal BoardNumber As Integer,
ByRef Outputs As Integer ,
ByRef Sens As Integer) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will do the opposite of an update on an analogue I/O slave. It will read back the current state of the Outputs and the Sensitivity settings on the board specified by Board Number. This is designed for the situation where the PC has been turned off or the application program has been terminated but the slave board remains on. It allows the PC to update its record of the outputs before resuming changes to them. Bits 0 to 7 of Outputs correspond to outputs 1 to 8 respectively with a '1' being "on". Bits 0 to 6 of "Sens" correspond to the sensitivity settings of analogue inputs 1 to 7 respectively with a '1' being "High Sensitivity"

Usage "C / C++":

int Outputs, Sens, Error;

BoardNumber = 10;

Error = CM_ANIO_GetStatus(BoardNumber, &Outputs, &Sens);

If(Error == 0)

{

Use Outputs and Sens // Outputs and Sens now contain valid data as read from board 10

}

Usage "Basic / Visual Basic":

Dim BoardNumber As Integer

Dim Outputs As Integer

Dim Sens As Integer

BoardNumber = 1

Error = CM ANIO_GetStatus(BoardNumber, Outputs, Sens)

If Error = 0 Then

Use Outputs and Sens ' Outputs and Sens now contain valid data as read from board 1

End If

Function Name: CM_Motor_M1Speed

Applicable to: Motor Slave

Syntax 'C/C++': int CM_Motor_M1Speed (int BoardNumber, int Speed, int *Inputs)

Syntax 'Basic/VB': CM_Motor_M1Speed (ByVal BoardNumber As Integer,
ByVal Speed As Integer,
ByRef Inputs As Integer) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will set the speed of motor 1 on the motor slave specified by "BoardNumber". The current board inputs will be returned in the "inputs" variable. Bits 0 to 7 of Inputs correspond to inputs 1 to 8 respectively with a '1' being "High" (i.e. "+5v").

Usage "C / C++":

int Inputs, Speed, Error;

BoardNumber = 6;

Speed = 127; // approximately half of full speed

Error = CM_Motor_M1Speed(BoardNumber, Speed, &Inputs);

If(Error == 0)

{

Use Inputs // Inputs now contains valid data read from board 6

}

Usage "Basic / Visual Basic":

Dim BoardNumber As Integer

Dim Inputs As Integer

Dim Speed As Integer

BoardNumber = 13

Speed = 255 ' maximum speed

Error = CM_Motor_M1Speed(BoardNumber, Speed, Inputs)

If Error = 0 Then

Use Inputs ' Inputs now contains valid data read from board 13

End If

Function Name: CM_Motor_M2Speed

Applicable to: Motor Slave

Syntax 'C/C++': int CM_Motor_M2Speed (int BoardNumber, int Speed, int *Inputs)

Syntax 'Basic/VB': CM_Motor_M2Speed (ByVal BoardNumber As Integer,
ByVal Speed As Integer,
ByRef Inputs As Integer) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will set the speed of motor 2 on the motor slave specified by "BoardNumber". The current board inputs will be returned in the "inputs" variable. Bits 0 to 7 of Inputs correspond to inputs 1 to 8 respectively with a '1' being "High" (i.e. "+5v").

Usage "C / C++":

int Inputs, Speed, Error;

BoardNumber = 16;

Speed = 64; // approximately a quarter of full speed

Error = CM_Motor_M2Speed(BoardNumber, Speed, &Inputs);

If(Error == 0)

{

Use Inputs // Inputs now contains the current inputs as read from board 16

}

Usage "Basic / Visual Basic":

Dim BoardNumber As Integer

Dim Inputs As Integer

Dim Speed As Integer

BoardNumber = 19

Speed = 255 ' maximum speed

Error = CM_Motor_M2Speed(BoardNumber, Speed, Inputs)

If Error = 0 Then

Use Inputs ' Inputs now contains the current inputs as read from board 19

End If

Function Name: CM_Motor_M1Direction
Applicable to: Motor Slave
Syntax 'C/C++': int CM_Motor_M1Direction(int BoardNumber, int Direction, int *Inputs)
Syntax 'Basic/VB': CM_Motor_M1Direction (ByVal BoardNumber As Integer,
ByVal Direction As Integer,
ByRef Inputs As Integer) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will set the direction of motor 1 on the motor slave specified by "BoardNumber". The value of "Direction" should be 1 for "Stop", 2 for "Forward" and 3 for "Reverse". Note: #defines for these constants can be found in the dll header file "masdll.h". The current board inputs will be returned in the "inputs" variable. Bits 0 to 7 of Inputs correspond to inputs 1 to 8 respectively with a '1' being "High" (i.e. "+5v").

Usage "C / C++":

```
int Inputs, Direction, Error;
```

```
BoardNumber = 27;
```

```
Direction = 2; // forward
```

```
Error = CM_Motor_M1Direction( BoardNumber, Direction, &Inputs );
```

```
If(Error == 0)
```

```
{
```

```
    Use Inputs // Inputs now contains the current inputs as read from board 27
```

```
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
```

```
Dim Inputs As Integer
```

```
Dim Direction As Integer
```

```
BoardNumber = 15
```

```
Direction = 3 ' reverse
```

```
Error = CM_Motor_M1Direction( BoardNumber, Direction, Inputs )
```

```
If Error = 0 Then
```

```
    Use Inputs ' Inputs now contains the current inputs as read from board 15
```

```
End If
```

Function Name: CM_Motor_M2Direction
Applicable to: Motor Slave
Syntax 'C/C++': int CM_Motor_M2Direction(int BoardNumber, int Direction, int *Inputs)
Syntax 'Basic/VB': CM_Motor_M2Direction (ByVal BoardNumber As Integer,
ByVal Direction As Integer,
ByRef Inputs As Integer) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will set the direction of motor 2 on the motor slave specified by "BoardNumber". The value of "Direction" should be 1 for "Stop", 2 for "Forward" and 3 for "Reverse". Note: #defines for these constants can be found in the dll header file "masdll.h". The current board inputs will be returned in the "inputs" variable. Bits 0 to 7 of Inputs correspond to inputs 1 to 8 respectively with a '1' being "High" (i.e. "+5v").

Usage "C / C++":

int Inputs, Direction, Error;

BoardNumber = 8;

Direction = 1; // stop

Error = CM_Motor_M2Direction(BoardNumber, Direction, &Inputs);

If(Error == 0)

{

Use Inputs // Inputs now contains the current inputs as read from board 8

}

Usage "Basic / Visual Basic":

Dim BoardNumber As Integer

Dim Inputs As Integer

Dim Direction As Integer

BoardNumber = 30

Direction = 3 ' reverse

Error = CM_Motor_M2Direction(BoardNumber, Direction, Inputs)

If Error = 0 Then

Use Inputs ' Inputs now contains the current inputs as read from board 30

End If

Function Name: CM_Motor_ReadInputs

Applicable to: Motor Slave

Syntax 'C/C++': int CM_MotorReadInputs (int BoardNumber, int *Inputs)

Syntax 'Basic/VB': CM_MotorReadInputs (ByVal BoardNumber As Integer,
ByRef Inputs As Integer) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will read the current inputs present on the relay slave specified by Board Number and return them in the variable "Inputs". Bits 0 to 7 of Inputs correspond to inputs 1 to 8 respectively with a '1' being "High" (i.e. "+5v").

Usage "C / C++":

```
int Inputs, Error;
```

```
BoardNumber = 17;
```

```
Error = CM_MotorReadInputs( BoardNumber, &Inputs);
```

```
If(Error == 0)
```

```
{
```

```
    Use Inputs           // Inputs now contains valid inputs as read from board 17
```

```
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
```

```
Dim Inputs As Integer
```

```
BoardNumber = 4
```

```
Error = CM_Motor_ReadInputs( BoardNumber, Inputs)
```

```
If Error = 0 Then
```

```
    Use Inputs           ' Inputs now contains valid inputs as read from board 4
```

```
End If
```

Function Name: CM_Motor_GetStatus

Applicable to: Motor Slave

Syntax 'C/C++': int CM_Motor_GetStatus (int BoardNumber,
int *M1Speed, *M1Direction,
int *M2Speed, *M2Direction,
int *Inputs)

Syntax 'Basic/VB': CM_Motor_GetStatus (ByVal BoardNumber As Integer,
ByRef M1Speed As Integer,
ByRef M1Direction As Integer,
ByRef M2Speed As Integer,
ByRef M2Direction As Integer,
ByRef Inputs As Integer,) As Integer

Return Value: Returns '0' if successful or a negative number that corresponds to the error. A list of possible errors can be found in the dll header file "masdll.h"

Description: This function will read back the current speed and direction of both motors as well as the current inputs from the board specified by Board Number. This is designed for the situation where the PC has been turned off or the application program has been terminated but the slave board remains on. It allows the PC to update its record of the current speeds and directions before resuming changes to them. Bits 0 to 7 of Inputs correspond to Inputs 1 to 8 respectively with a '1' being "on".

Usage "C / C++":

```
int M1Speed, M1Direction, M2Speed, M2Direction, Inputs, Error;
```

```
BoardNumber = 29;
```

```
Error=CM_Motor_GetStatus(BoardNumber, &M1Speed, &M1Direction, &M2Speed, &M2Direction,  
&Inputs);
```

```
If(Error == 0)
```

```
{  
    // variables now contain valid data as read from board 29  
    Use M1Speed, M2Speed, M1Direction, M2Direction, Inputs  
}
```

Usage "Basic / Visual Basic":

```
Dim BoardNumber As Integer
```

```
Dim M1Speed As Integer
```

```
Dim M1Direction As Integer
```

```
Dim M2Speed As Integer
```

```
Dim M2Direction As Integer
```

```
Dim Inputs As Integer
```

```
BoardNumber = 2
```

```
Error = CM_Motor_GetStatus( BoardNumber, M1Speed, M1Direction, M2Speed, M2Direction, Inputs)
```

```
If Error = 0 Then
```

```
    ' variables now contain valid data as read from board 2  
    Use M1Speed, M1Direction, M2Speed, M2Direction, Inputs
```

```
End If
```

9. Minimum PC System Requirements

The Control Master range of boards and associated software do not require a high spec PC for correct operation, but the following system is suggested as a sensible minimum

Processor	500MHz Pentium
Memory	64MB
HDD	10MB free space required
Screen Resolution	1024x768 (256 colours)
Interface	One free USB socket (1.0 or 2.0)
Operating System	Windows XP or Vista

WARNING: The control master adaptor board range is intended for DC voltages less than 50v. It should not be connected directly to mains voltages under any circumstances.

Terms of Use for all Goods Supplied

Definitions

'Supplier' shall mean PC Control Ltd.

'Buyer' shall mean the person, company or any other body that purchases or agrees to purchase Goods.

'Goods' shall mean all goods and services which the Buyer agrees to buy from the Supplier including replacements for defective Goods, hardware, documentation and software products licensed for use by the Buyer.

Use of the Goods in any way by the Buyer constitutes acceptance of these terms and conditions.

Terms and Conditions

1. The Goods are intended to be part of the buyer's own design of apparatus and not a finished product in their own right.
2. The Goods supplied are not to be used in any design where there is a risk, however small, either directly or indirectly, of death or personal injury.
3. The Buyer will be responsible for ensuring the fitness for purpose of the Goods for the Buyer's application.
4. To the extent permitted by law, the Supplier accepts no liability whatsoever or howsoever arising in respect of loss, damage or expense arising from errors in information or advice provided whether or not due to the Supplier's negligence or that of its employees, agents or sub-contractors save for any loss or damage arising from death or personal injury.
5. To the extent permitted by law, the Supplier shall not be liable to the Buyer by reason of any representation (unless fraudulent), or any implied warranty, condition or other term, or any duty at common law, or under the express terms of any Contract with the Buyer, for any indirect, special or unforeseen loss or damage (whether for loss of profit or otherwise), costs, expenses or other claims for compensation whatsoever (whether caused by the negligence of the Supplier, its employees or agents or otherwise) which arise out of or in connection with the supply of the Goods or their use or resale by the Buyer.
6. The entire liability of the Supplier under or in connection with the Contract with the Buyer shall not exceed the price of the Goods except as expressly provided in these terms and conditions.
7. These terms are an important part of the full terms and conditions of business as published on the website at www.pc-control.co.uk/general-terms.htm which also apply.

If you cannot agree to the terms and conditions of use of the control master board range or software then you should return it to the supplier within 7 days of receipt to receive a refund. Your use of the boards or the associated software in any way whatsoever will be regarded as an acceptance of these terms and conditions.