Mclennan Servo Supplies Ltd

PM600 Motion Controller
USER Manual

B

PRODUCT MANUAL FOR PM600 Digiloop ®

SAFETY NOTICE!

Position control systems are inherently hazardous. Even a small motor, if coupled to a leadscrew, gearbox, or any other form of mechanism that provides a mechanical advantage, can generate considerable force and could cause serious injury. Incorrect operation can also lead to damage to the motor or associated machinery. It is essential that the purpose of the various fault detection features built into the PM600 be fully understood and used correctly.

Caution

STATIC SENSITIVE DEVICES

This unit has static sensitive devices. Observe handling precautions: Hold card by edges only. Do not touch components or connector pins. Ship only in anti-static packaging.

Mclennan Servo Supplies Ltd.
Unit 1, The Royston Centre,
Lynchford Road, Ash Vale,
UK GU12 5PQ

Telephone:	+44 (0)8707 700 700
FAX:	+44 (0)8707 700 699
Sales EMAIL:	sales@mclennan.co.uk
Technical support EMAIL:	tech@mclennan.co.uk

This manual is written for firmware version – V5.19a

The manufacturer reserves the right to update the data used in this manual in line with product development without prior notice.

This manual printed: Thursday, 15 September 2005

CONTENTS

1 Introduction

The PM600 *Digiloop®* controller is a serial interfaced, Eurocard format, intelligent motor controller. It can be used in systems with DC brushed servo motors, AC brushless servo motors or stepper motors. It can be run in open or closed loop modes. In closed loop modes it utilises encoder feedback to continually monitor position, speed and direction. In most servo systems no tacho-generator is necessary.

The minimum arrangement for a PM600 system consists of either a stepper motor or a servo motor fitted with an incremental encoder (encoder not needed for open-loop stepper mode) a servo amplifier or a stepper drive, a PM600 controller and suitable system power supplies.

1.1 Features

The PM600 allows the user to control motor position, velocity, acceleration and deceleration either using commands in ASCII format, sent down an RS232 or RS485 data path or via manual controls.

The PM600 can be set-up to operate as either a servo motor position controller or a stepper motor position controller.

The manual control inputs can be *Jog* push-buttons, a joystick or quadrature encoder type signals from a device such as a trackerball.

In Servo mode the PM600 can be used to synchronise the position of its motor to another quadrature signal at user defined ratios. This is known as *electronic gearbox* mode. These quadrature signals can come from an encoder on another motor or an encoder fitted to the mechanism.

Up to 99 PM600 controllers can be daisy-chained along an RS232 data bus or connected in parallel along an RS485 multi-drop data bus. A number defining the axis to be addressed prefixes each command. A PM600 will pass on all commands and only act upon a command prefixed by its own address.

Strings of such commands can be sent directly from a host computer or processor in immediate *real time* mode, or loaded into the on-board *flash* memory of the PM600 and executed in sequence. In either mode the controller can be interfaced to external devices via its I/O facilities to take account of various contingencies, and provide a handshake with other machine functions.

An on board switched-mode regulated power supply allows the PM600 to be supplied from an unregulated DC source of between +10V and +32V.

Two types of limits are provided; *hard* limits and *soft* limits as well as a stop input.

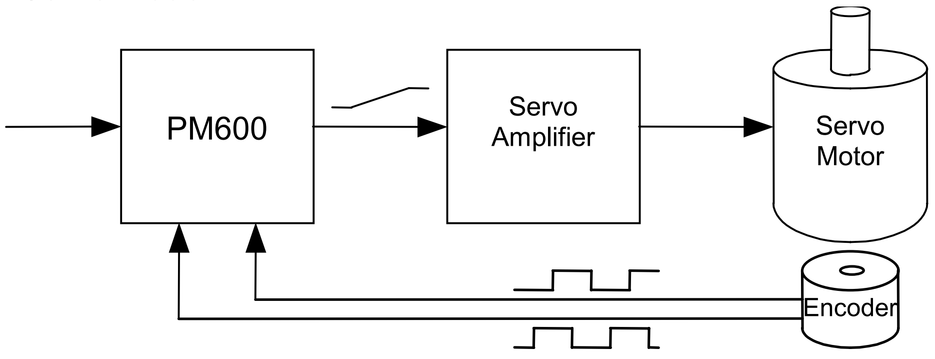
2 Operation

A memory jogging list of the commands available and their function can be found by using the Help (**HE**) command for the first page and Help Next (**HN**) command for further pages.

2.1 Operating Modes

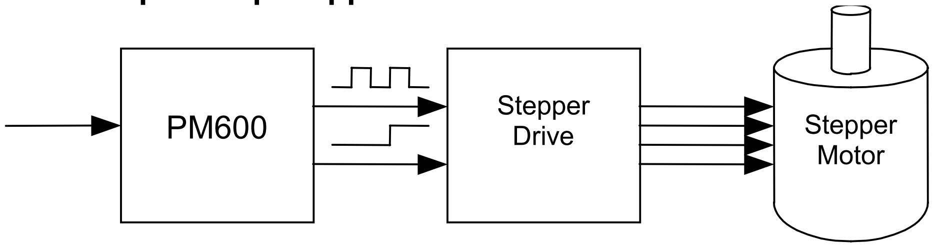
The PM600 can be set-up to operate as either a servo motor position controller or a stepper motor position controller. In the stepper mode of operation there are four sub modes of operation, Open-loop mode, Checking stepper mode, External loop stepper mode, Closed-loop stepper mode.
The mode of operation of the PM600 is set by the **CM** command.

2.1.1 CM1 Servo Mode



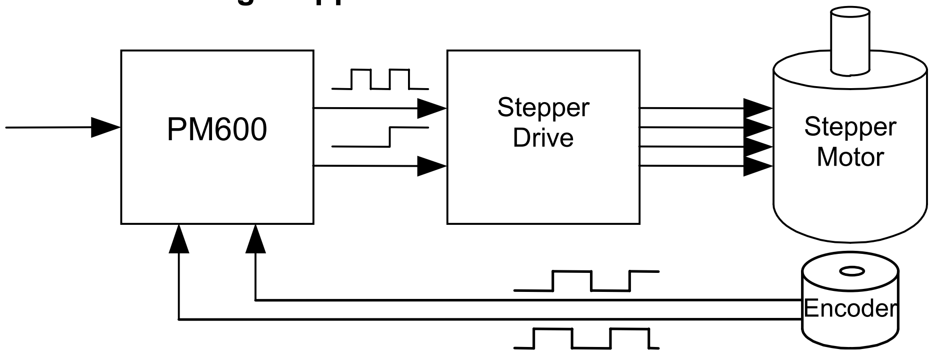
The system consists of servo motor fitted with an incremental encoder, a servo amplifier and a PM600 controller.
The PM600 generates a *command* position and compares it with the *actual* position read from the incremental encoder. Any error between these two is used by the PM600 to calculate an error voltage that is feed to a servo amplifier. The servo amplifier drives the servo motor that drives the load to the correct position.

2.1.2 CM11 Open loop stepper mode



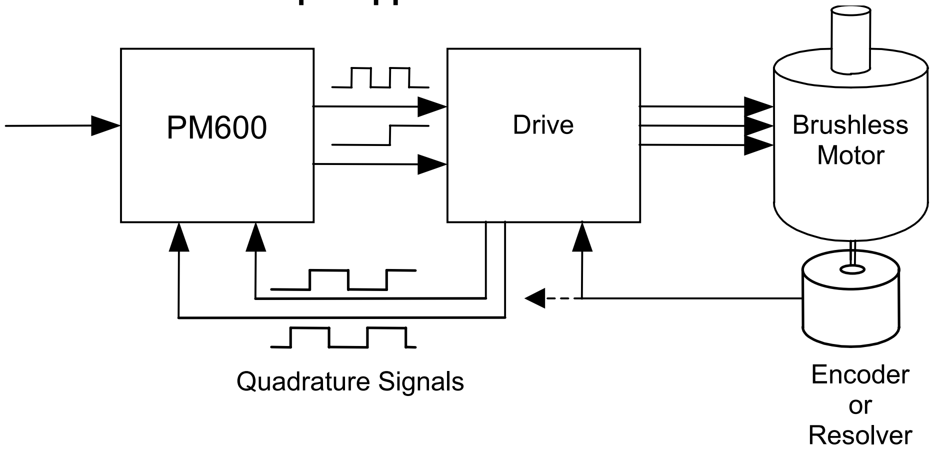
The system consists of stepper motor, a stepper drive *translator* and a PM600 controller.
The PM600 controller receives move command signals from the RS232 port, manual *jog* signals or from stored sequences. It will then generate the required direction signal and clock (step) pulses that are fed to a stepper drive. The drive will then convert these into the drive currents to drive the stepper motor.

2.1.3 CM12 Checking Stepper Mode



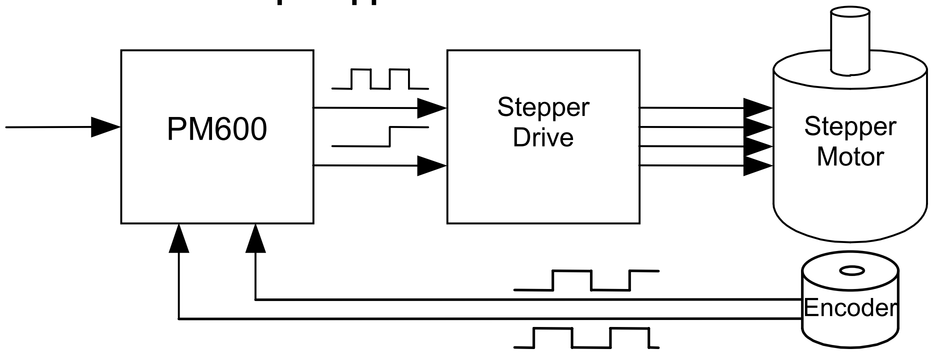
The system consists of stepper motor fitted with an incremental encoder, a stepper drive *translator* and a PM600 controller.
As open-loop, but with an incremental encoder fitted to the motor or mechanism to monitor its actual position. Quadrature signals from the encoder are fed to the PM600 to allow the move to be checked. Any error is not corrected.

2.1.4 CM13 External loop stepper mode



The system consists of brushless motor fitted with rotation sensor, a drive and a PM600 controller.
As open-loop but with the drive correcting for any position errors. The actual position of the motor is monitored the encoder or resolver fitted to the motor. Quadrature signals generated by the drive or encoder are fed to the PM600 to allow the motor position to be monitored.

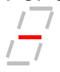
2.1.5 CM14 Closed loop stepper mode

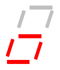


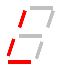
The system consists of stepper motor fitted with an incremental encoder, a stepper drive *translator* and a PM600 controller.
As open-loop but with an incremental encoder fitted to the motor or mechanism. The actual position is monitored during a move to detect a motor stall and the end of move criterion. Quadrature signals from the encoder are fed to the PM600 to allow the move to be checked. Any error is automatically corrected.

2.2 Front panel Status Display

A red seven-segment LED display marked 'STATUS' is mounted behind the front panel. This display will show various states and operations that PM600 is executing.
For example:

 Idle

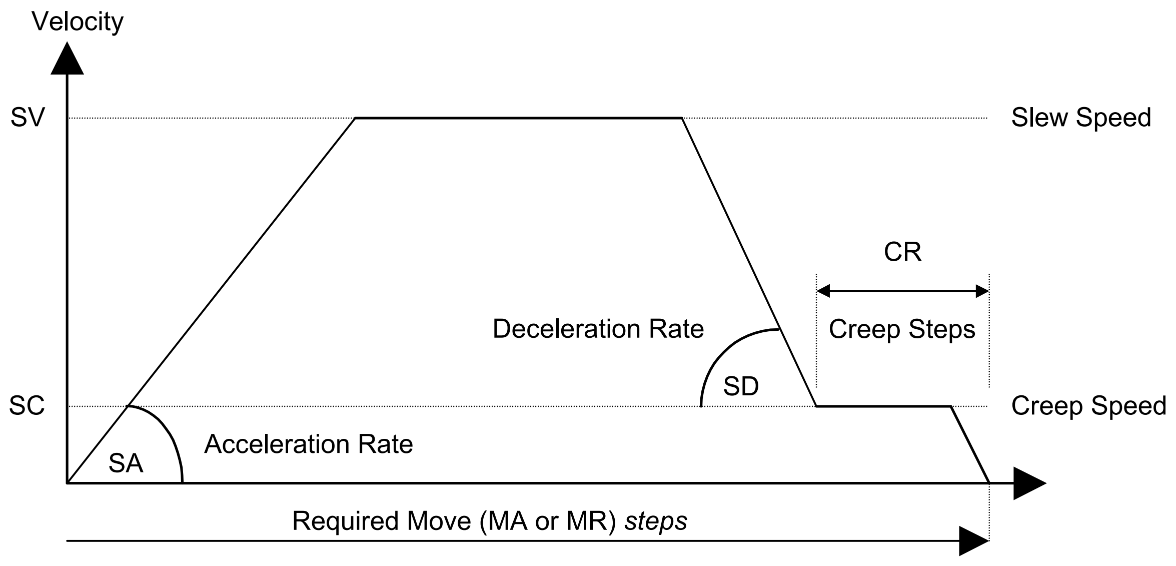
 Move

 Lower limit activated

Refer to Status Display section for more information.

2.3 Typical Servo Movement Profile

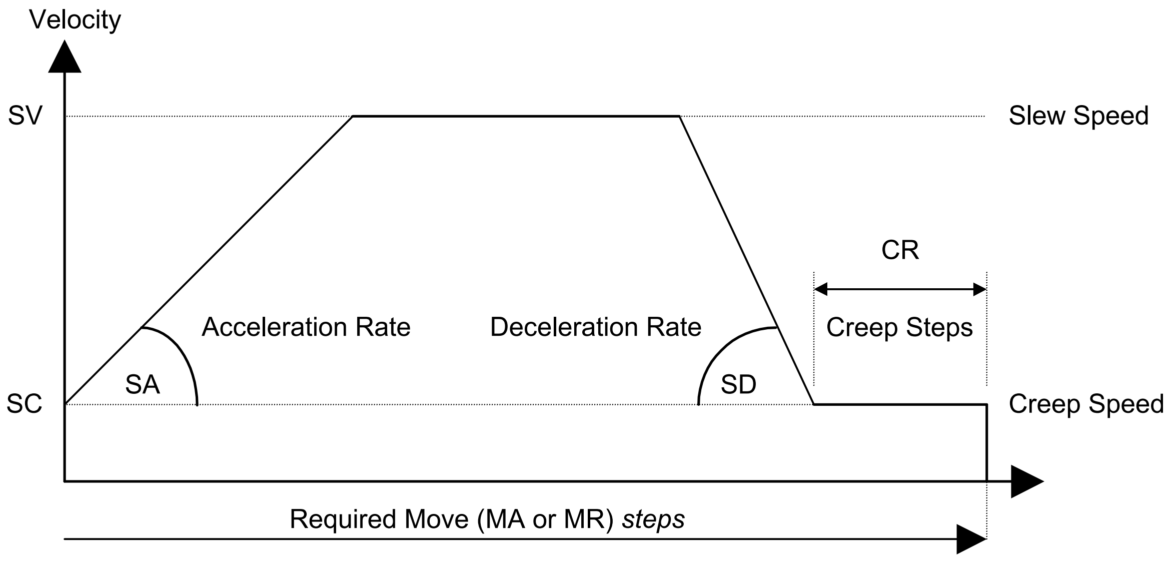
Either the MA (move absolute) or MR (move relative) command executes a typical move using the PM600 controller. The motor speed will ramp up linearly at the rate defined by the **SA** (set acceleration) command, until the *slew speed* is reached (programmable by the **SV** command). The motor will continue at this speed until it decelerates at the **SD** rate, and then finish the last steps at the *Creep Speed* defined by the **SC** (set creep speed) command. The number of these last *Creep steps* is defined by the **CR** command. Note that this move profile is applied to the command position.



The end of a move is defined as having occurred when the actual position has settled close to the required position. The distance at which the motor starts to *settle* is set by the **WI** (end of move window) command. For the move to be defined as being complete, the motor must *settle* within the end of move window for the time set by the **SE** (settle time) command.

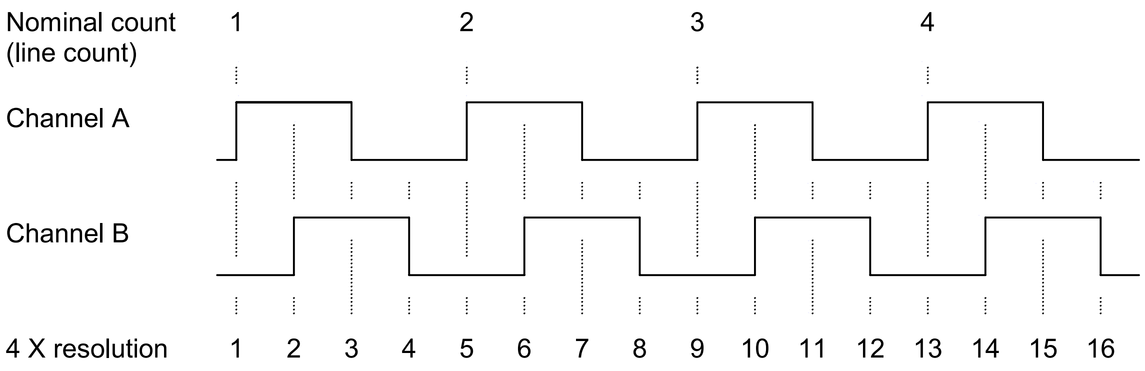
2.4 Typical Stepper Movement Profile

Either the MA (move absolute) or MR (move relative) command executes a typical move using the PM600 controller. The motor speed will ramp up linearly at the rate defined by the **SA** (set acceleration) command from the *creep speed* (defined by **SC** command). When the *slew speed* is reached (programmable by the **SV** command) the motor will continue until it decelerates at the **SD** rate, and then finish the last steps at the *creep speed*. The number of these last *creep steps* is defined by the **CR** command.



2.5 Encoder

Using a PM600 it is possible to monitor the motor or mechanism position with an incremental shaft encoder. The nominal resolution of the encoder is multiplied by four since all transitions of the quadrature signals are counted:



2.5.1 Encoder Types

The PM600 can be used with encoder producing either 5V TTL signals or 5V differential signals. Other terminology for differential signals is RS422, line driver and complementary pair.
If using differential signals then the encoder termination switches should be switched on. With 5V TTL, open-collector or single-ended encoders, the encoder termination switches should be switched off. Refer to the Encoder Termination section for details.
Using a single-ended encoder, speeds of up to 200 K steps/sec can be obtained. Using an encoder that has differential outputs up to 1.2 M steps/sec can be obtained.

2.5.2 Clock Rate

The encoder inputs go through a four stage digital filter clocked by a 4 MHz clock. This means that the minimum pulse width presented to the encoder inputs must be greater than 1.5 µs.

2.6 Command Word Syntax

The PM600 has a wide range of command options extending beyond the main move functions. The aim is to provide a flexible and comprehensive control device for integration of motion control into larger systems.

2.6.1 Commands

Most commands are two letters, the function of each, being described in the Programmer Reference section. Each command is preceded by the appropriate address to identify the axis for which the command is intended.
Where applicable (e.g. move commands, setting of system parameters, etc.) the command should then be followed by the desired value:

aXXnnn<cr>

a = address
XX = command
nnn = value (if required)
<cr> = carriage return.

Command strings should be terminated with a carriage return character (ASCII 0D hex).
Upper or lower case characters can be used for the command. Spaces within the command line are ignored.
If no value is given, then zero is assumed.

All commands except for **Control C** and **ESCAPE** are buffered. Commands are executed in consecutive order. Commands will be acted on sequentially, as they have been entered. If any command cannot be executed immediately (because it may need to wait for some condition or a previous command to finish) then the command and any that follow it will be buffered internally (up to 256 characters). The responses for each command are returned as the command is executed.

NOTE: the controller does NOT detect Delete, backspace and cursor movement characters. With some terminals or emulators these keystrokes will be translated as an escape sequence, i.e. a sequence of characters beginning with an escape character (ASCII 1B hex). The controller will detect the escape character and act on it accordingly.

Please note that due to the loop nature of the RS232 communications, all characters that are sent to the controllers are echoed back to the host.

2.6.2 Replies

Response to a command, once it has been accepted, is either an **OK** string or an alpha-numeric string. Responses terminate in a carriage return character (0D) and a line feed character (0A) and are preceded by the address number (always two characters) and a colon character. An appropriate message is sent if a mistake or conflicting instruction creates an error. The first character of an error message (after the address prefix) is **!**

2.7 Non-volatile Memory

All set-up parameters (control coefficients, acceleration, deceleration, velocities, jog speeds, creep speed, etc.), sequences and profiles will be read from the on board FLASH memory to the controllers normal RAM memory on power-up. The parameters can then be modified in the volatile RAM by the relevant commands, but these modifications will not persist after power down. Use the backup (BA, BC, BD, BP & BS) commands to write the current set-up back to the non-volatile FLASH memory, so that they will be ‘remembered’ on power up.
Important – if the set-up of the PM600 is changed then the **BD** (**B**ackup **D**igiloop) **must** be executed to save the set-up values to Flash memory. If the set-up values ore not saved to Flash memory, the values will be lost on power-down.

2.8 Hardware Fault Detection Inputs

2.8.1 Abort Stop

The *abort stop* input, if triggered will stop any further movement of the axis. Using the abort mode command this input can either be latched until the controller is reset or automatically reset when the *abort stop* input is closed.

The front panel STATUS display will also show indicating Stop Input Abort. If you attempt a move on axis 1 by a **1MR10** command, the controller will respond with a **01:! INPUT ABORT** error message.
Sending a **CO** (current operation) command will check the state of the *input abort*. If the Stop input has been activated on axis 1, a response of **01:Input Abort** will be received. This condition is reset with the **RS** command.

2.8.2 Hard Limits

There are *hard limit* inputs that can be connected to end of travel switches to prevent any further movement in their direction.

The status of the *hard limits* can be found using the **OS** (output status) command. This command will give a response of a string of 8 numeric characters of either 0 or 1. This binary string will represent the current status of the controller.
If the communications are in Verbose Mode, the reply is preceded by **Status = .** The two bits that show the status of the *hard limits* are **c** and **d**.

Status = abcdefgh	where:	
c -	0 –	Upper hard limit is OK
	1 –	Upper hard limit is activated
d -	0 –	Lower hard limit is OK
	1 –	Lower hard limit is activated

Example:

If the controller of axis 1 is currently stopped on the upper hard limit:

1OS	in Verbose Mode will respond:	**01:Status = 10100000**
1OS	in Quiet Mode will respond:	**01:10100000**

The status of the *hard limits* can also be seen on the **QA** page.

Whilst either *hard limit* is activated, the front panel STATUS display will show if the Upper *hard-limit* (or both limits) is activated and if the Lower *hard-limit* is activated.

The Abort Stop input and Limit inputs must not be used as a safety device or part of a safety system for ensuring the safety of persons

2.9 Software Fault Detection Functions

2.9.1 Soft Limits

The *soft limits* can be used to prevent any movement outside of a programmable range. This range is only relative to the zero point.
The command to set the Upper Soft Limit Position is **UL** and the command to set the Lower Soft Limit Position is **LL**. Subsequent moves by the Move Absolute (**MA**), Move Relative (**MR**), Constant Velocity (**CV**) or manual Jog moves will not be allowed above this Upper Limit. If an attempt to set either the upper limit below or equal to lower limit or to set the lower limit above or equal to upper limit a **! LIMITS CONFLICT** error message will be received.

Examples:

1UL8000	Set the axis 1 Upper Soft Limit Position to 8000.
1LL-4000	Set the axis 1 Lower Soft Limit Position to -4000.

The **SL** command is used to enable or disable the soft limit protection. If the soft limits are disabled, further movement is NOT bounded by the upper and lower soft limits. Hard limits will still be active and cannot be disabled.

Example:

1SL0	Sets the soft limits OFF (disabled) for controller axis 1.
1SL1	Sets the soft limits ON (enabled) for controller axis 1.

2.9.2 Stall

In servo control mode, a stalled motor (or encoder failure) is detected by looking for changes in the position encoder signals (or equivalently the changes in observed motor position). If the motor does not move, and the voltage output value from the PM600 exceeds the value set by the **TH** command for a time of 256ms, then the PM600 will set its output to zero and set Stall Abort condition. The threshold is expressed as a percentage of full scale output of the Analogue output.
Failure of an encoder is indistinguishable from a stalled motor, and messages from the PM600 refer to *stall abort* rather than encoder failure.
The servo system will have coulomb friction and the voltage required to overcome this friction, varies from system to system, so the value of **TH** must be large enough not to nuisance trigger but small enough to detect any failure.

If a *stall abort* condition occurs, the front panel status display shows a , and movement is stopped. Subsequent moves will not function but will return the response **! STALL ABORT** until reset by either a Reset (**RS**) command or by powering off. The stall abort function can be enabled or disabled by using the **AM** (abort mode) command.

Caution: Do not disable fault detection features unless sure of operation.

The response to a **CO** (Current Operation) command is **! STALL ABORT**.

Example:

1TH40	Set the Threshold before *motor stalled* condition for axis 1 to 40%.

Note that if a stepper motor stalls, this condition will be detected as a tracking error.

2.9.3 Tracking

The *Tracking window* is the allowable difference between the *Command Position* and the *Actual Position*. The **TR** command is used to set the tracking window. When the motor is stationary this is the allowable static error. During a move, a changing *command position* is generated. The *Tracking Window* operates on the difference between the *actual position* and this moving *command position*. The servo system will have a *following error,* so the value of **TR** must be large enough not to nuisance trigger but small enough to detect any failure.

If the *tracking* window is exceeded, the front panel display will show a (tracking error). If the *tracking abort* is enabled the controller will *abort* and the *error output* signal will be activated.
This abort function can be enabled or disabled by using the **AM** (abort mode) command.

Caution: Do not disable fault detection features unless sure of operation.

If aborted, subsequent moves will not function but will return the response **! TRACKING ABORT** until reset by either a Reset (**RS**) command or by powering off.

Example:

1TR400	Set the Tracking Window for axis 1 to 400 steps.

2.9.4 Time-out and Not Complete

The maximum time allowed at the end of a move, from when the Command Position reaches its target, until the move has settled and completed. The **TO** command will set the Not Complete/Time-Out time. If either the move or the error correction is not completed within this time then a Time-out will be detected and the controller will *abort.*

This abort function can be enabled or disabled by using the **AM** (abort mode) command.

Caution: Do not disable fault detection features unless sure of operation.

If aborted, subsequent moves will not function but will return the response **! NOT COMPLETE/TIMEOUT ABORT** until reset by either a Reset (**RS**) command or by powering off.

Example:

1TO4000	Set the axis 1 Time out to 4 seconds (4000mS).

2.9.5 Abort Mode

The Abort Mode (**AM**) command can be used to set the conditions that cause an abort and disable the control (servo) loop.
The syntax of the command is:
AMabcdefgh

where	**a** -	0 – Abort Stop Input disables control loop
		1 – Abort Stop Input stops all moves only
	b -	0 –Abort Stop Input is latched requiring RS command to reset
		1 – Abort Stop Input is only momentary
	c -	0 – Stall Error disables control loop
		1 – Stall Error is indicated but control loop remains active
	d -	0 – Tracking Error disables control loop
		1 – Tracking Error is indicated but control loop remains active
	e -	0 – TimeOut Error disables control loop
		1 – TimeOut Error is indicated but control loop remains active
	f -	Reserved for future use.
	g -	Reserved for future use.
	h -	0 – Enable output switched OFF during a disabled control loop
		1 – Enable output left ON during a control loop abort

Caution: Do not disable fault detection features unless sure of operation.

Examples:

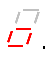
1AM00010001	Set axis 1 to abort on all conditions except Tracking Error, enable output stays ON.
1AM11000000	Set axis 1 to abort on all conditions except momentary Abort Stop input only stops moves.

2.10 Move Commands

2.10.1 Move Distance

There are two commands that are used to move the motor a set distance. The command **MA** (Move Absolute) will execute a move to a position relative to the *Command Position* of zero. The command **MR** (Move Relative) will execute a move to a position relative to the current *Command Position.*
The range of moves that can be made are in the range –2147483647 to 2147483647 (±2^{32}). If the move command cannot be accepted one of the following error messages will be received.

! HARD LIMIT	Hard limit for required direction is already activated
! SOFT LIMIT	Move attempted that exceeds the Soft limit in the required direction
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A time-out abort has been detected

During a move the front panel display will show a . The response to a **CO** (Current Operation) command while an **MA** or **MR** command is taking place is **01:Move**.

Examples:
If axis 1 has a current Command Position of 5000 then the command
1MA4000 Will move 1000 steps in the negative direction to arrive at a Command position of 4000.
The command
1MR4000 Will move 4000 steps in the positive direction to arrive at a Command position of 9000.

2.10.2 Constant Velocity Move

A *Constant velocity* move is used to move continuously at the required speed. The command to start a constant velocity move is **CV**, with an argument which is the value of the required speed in steps/sec. Initially the move will ramp up at the rate set by the set acceleration (**SA**) command to the speed given in the argument. The polarity of the value dictates the direction of movement. Subsequent CV commands can be sent to change the required velocity, including changes in direction.
The speed is changeable whilst motion is in progress. The **SA** and **SD** rates define the rate at which the change of speed will be made. An ST command, ESCAPE or Control C exits constant velocity mode.
The soft limits are active in CV mode. For continuous applications they must be disabled with the SL command.
The range of velocities that can be used are (dependant on the system) are–1200000 to 1200000.

During a move the front panel display will show a . The response to a **CO** (Current Operation) command while a **CV** command is taking place is **01: Constant velocity**.

Examples:

1CV2000	Start constant velocity move in positive direction at 2000 steps/sec on axis 1.
1CV-10000	Start constant velocity move in negative direction at 10,000 steps/sec on axis 1.

2.10.3 Creep Distance

It may be important to travel the last part of a move at Creep speed to avoid position overshoot. The number of Creep Steps is set by the **CR** command in steps. This is the number of steps that will be moved at the Creep Speed. At the end of each move the motor will decelerate and execute this number of steps at the creep speed. To ensure that the creep speed is reached at the end of the deceleration phase, the number of creep steps is subtracted from the move distance and then the deceleration point is calculated.

The response to a **CO** (Current Operation) command while the Creep steps are taking place at end of move is **01:Creep**.

Example:

1CR50	Set the creep distance to 50 steps on axis 1.

2.10.4 Backoff

A number of back-off steps can be set to execute at the end of each move by the **BO** (back-off) command. This can be useful in combating backlash in a mechanism. The controller will therefore always approach the final position at the creep speed and from the same direction.

The motor will decelerate to the creep speed at the back-off position relative to the required end position. The controller will then complete the move at the creep speed

The response to a **CO** (Current Operation) command while the back-off is taking place at end of move is **01:Backoff**.

Example:

1BO500	Set the back-off distance to 500 steps on axis 1.

2.10.5 Gearbox Move Relative

When in gearbox mode, the **GM** (Gearbox Move Relative) command can be used to superimpose a relative move on top of the gearbox slaving. In this way, a correction in the synchronism of the two positions can be changed without exiting the gearbox mode. This move is done at the creep speed.

The response to a **CO** (Current Operation) command while this move is taking place is **01:Gearbox**.

Example:

1GM100	Superimpose a move of 100 steps (positive) on top of gearbox mode.

2.10.6 Home To Datum

The **HD** command is used to find a datum point of a mechanism. This is a special type of move that uses hardware inputs to define the end of the move.
Refer to the Datum Search section of this manual and the Datum Mode DM command for details on datum search use.
The **HD-1** command will perform the search in the negative direction.
Soft limits are **not** used during a Home to Datum search.

The response to a **CO** (Current Operation) command while this move is taking place is **01:Home to datum**.

Examples:

1HD	Search for datum point of axis 1 in positive direction.
1HD-1	Search for datum point of axis 1 in negative direction.

2.10.7 Move To Datum Position

If a *datum position* has already been captured, the **MD** (Move to Datum) command can be used to move the motor to the *datum position*

Example:

If axis 1 has a current valid Datum Position of 12496 then the command:
1MD Will move to the position of 12496.

2.10.8 Other Commands

The Delay (**DE**) command will initiate a delay counter with the time specified in milliseconds. After this delay the controller will return to the *Idle state*. Following moves and other commands that need an *Idle state* will therefore be postponed for the duration of the delay. For other commands that do not require an *Idle state*, such as Read Port or Write Port can be delayed by a Wait End command following the Delay command.

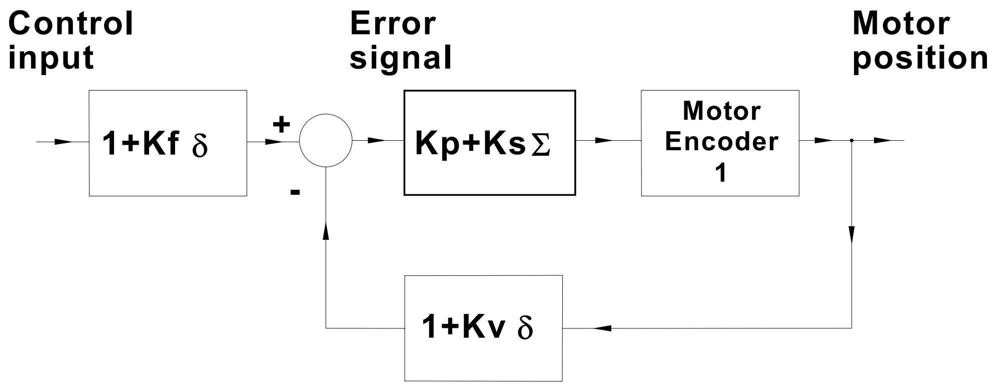
The response to a **CO** (Current Operation) command while this command is taking place is **01:Delay**.

The Wait End (**WE**) command will suspend further commands until the controller is in the *Idle state*. E.g. waiting for the end of a move or delay before operating a Write Port. Any commands that are sent to the PM600 will be buffered and executed after the move or delay is complete.

2.11 Servo Mode

This mode is selected by the command **CM1**. This is the default mode of the PM600.

2.11.1 Single Encoder Servo Mode



PM600 Single Encoder Coefficient Model

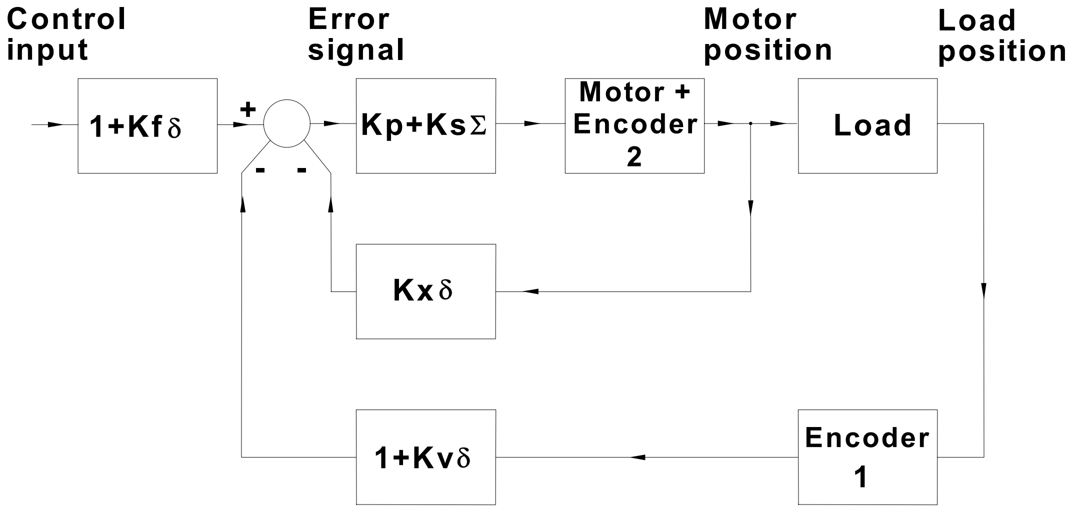
The above diagram illustrates the relationship between the control coefficients in a PM600 servo loop. The *control input* is a number generated by a move command (command position).

The PM600 then generates a signal to drive the motor via a servo amplifier. The encoder (usually mounted on the rear of the motor) produces a feedback signal of the motor's position (actual position). This enables the PM600 to calculate a position error signal and continuously update the command signal to the amplifier.

The PM600 in servo mode can be considered as a discrete-time *P.I.D.F.* controller (Proportional, Integral, Derivative & Feedforward). Coefficients can be varied to change the system characteristics or to optimise the response of the motor in a particular application.

2.11.2 Dual Encoder Mode

In high resolution systems where a remote encoder with a large number of counts per revolution of the motor is used, the amount of damping available from the **KV** coefficient may be insufficient to compensate for the mechanical backlash. Typical dual encoder systems use a rotary encoder fitted to the rear shaft of the motor and a high resolution encoder on the load.
An extra encoder on the rear shaft of the motor can be used to give the required damping factor. The level of the feedback signal from this encoder is controlled by the coefficient **KX**. The position (remote) encoder is connected to *Encoder 1* input and the motor encoder is connected to the *Encoder 2* input. The increased resilience of the coupling and the reduction in backlash between Encoder 2 and the motor, compared with that of Encoder 1 and the motor means that the value of **KX** can be much larger than **KV**.



PM600 Dual Encoder Coefficient Model

2.12 Stepper Modes

The mode of operation is set by the control mode command (**CM**). The options are as follows.

CM11	Open loop stepper mode
CM12	Checking stepper mode
CM13	External loop stepper mode
CM14	Closed loop stepper mode

There are various commands associated with control modes that have different actions according to the mode selected.

KP Set the proportional gain for auto-correction moves. The amount of attempted correction for each iteration is expressed as a percentage. The value is the difference between the Command Position and the encoder read Actual Position, scaled by **KP**. If the result is less than one step, then one step of correction will be used.

OF Output following error between current command and actual positions.

RS Reset. This command will reset the *tracking abort, stall abort, time out abort* or *user (command) abort* conditions and the enable output. It will also set the Command position to be equal to the Actual position (except open-loop stepper mode).

SE Set the settling time for all following moves. (The settling time operation depends on the PM600's control mode).

TO Set the time-out/not complete time. This is the maximum time allowed at the end of a move, from when the Command Position reaches its target, until the move has settled and completed. If the error correction is not completed within this time then a Time Out will be detected and Abort if set using the Abort Mode (**AM**) command.

WI Set Window. This command will set the acceptable error window for end of move checking. The *window* is the maximum acceptable difference between the *actual position* and the *command position* at the end of a move. The use of the 'end of move window' depends on the control mode.

2.12.1 Open-Loop Mode

CM11 Open-loop Stepper mode
Open-loop mode is the simplest mode where the motor is expected to move to the required position without extra feedback. In the open-loop stepper mode, if an encoder is fitted to the motor or mechanism the facility to monitor the actual position can be used. It is useful to run a stepper system in open-loop mode to check the operation of the encoder before changing to a closed-loop mode.
The settling time elapses either between the end of each move and the next or between the end of a move and the controller entering its *idle* state. The settling time is set by the **SE** command. The end of the previous move is simply the end of the move defined by the command position profile. It is used to allow for mechanical oscillations to cease. The end of move *window* (**WI**) not used in open-loop mode.

2.12.2 Closed-loop Modes

The position error checking functions of the PM600 can be invoked by using the closed-loop control modes. The Command Position will then be made the same as the Actual Position. It is important that the one to one relationship between Command position and Actual Position (excluding backlash) has been set up (see the ER command).

In closed-loop modes, if the difference between the Command Position and the Actual Position exceeds the Tracking Window, then the controller will detect that a *tracking error* has occurred. The Tracking Window is therefore used to set the allowable difference before a *tracking error* is detected. It is set by the **TR** command in steps. The value of **TR** should be set to a sufficient size to prevent false triggering e.g. by backlash if the encoder is mounted remotely from the motor.

When a *tracking error* is detected, all moves are stopped immediately and the *tracking error* condition is latched. Any further attempted moves or setting of certain parameters will not be executed and will return a `! TRACKING ABORT` error message. The error condition can be reset with the **RS** command, after investigating the cause.

CM12 Checking stepper mode

At the end of a move, the **SE** (settling time) counter counts down The settling time elapses at the end of each move to allow the motor to settle before the 'move complete' test occurs. The move is defined as being 'complete' if the **OF** (following error or position difference) value is less than the **WI** (end of move window) value. After the settling time reaches zero, if the *actual position* is within the **WI** range of the *command position*, the controller will either accept the next command or go to the *idle* condition. If the *actual position* is outside the *window* the response to any further move commands will be a **! NOT COMPLETE/TIMEOUT ABORT** message. The not complete condition can be reset with the **RS** command.

CM13 External loop stepper mode

This mode is for use with an external position correction device or a servo amplifier controlled by clock and direction signals. Automatic corrections will not be executed by the PM600 as the external device will be attempting to do this, therefore the ends of moves (**MA** or **MR**) will be handled differently from other closed-loop control modes.

At the end of a move, when the *actual position* comes within the **WI** range of this final target, the **SE** (settling time) counter counts down. When the settling time reaches zero the controller will either accept the next command or go to the *idle* condition.

If the Position overshoots the window before to the settling time reaches zero, the settling time counter is reset and started again. This means that for a move to be declared complete, the position difference (between *actual position* and *command position*) must be within the *window* for at least the *settling time*.

The external correction is only allowed to continue for a time defined by the Time Out parameter (set by the **TO** command in milliseconds). If the correction is not within the Window after this time has elapsed, then a move *Not Complete* condition will be declared.

The response to any further move commands will be a **! NOT COMPLETE/TIMEOUT ABORT** message. The not complete condition can be reset with the **RS** command.

CM14 Closed loop stepper mode

At the end of a move, the **SE** (settling time) counter counts down. The settling time elapses at the end of each move to allow the motor to settle before the 'move complete' test occurs. The move is defined as being 'complete' if the **OF** (following error or position difference) value is less than the **WI** (end of move window) value. If when the settling time reaches zero and the *actual position* is within the **WI** range of the *command position*, the controller and will either accept the next command or go to the *idle* condition.

However, if at the end of a move the *actual position* is outside the *window* a then a correction move will take place. A routine will start to attempt to correct the error between the Command and Actual positions. The difference between the two is scaled by the proportional gain coefficient (**KP**) and this value is the amount is the size of the correction move for this iteration. If the value is less than 1 step, then the correction move will be 1 step. The move will be made in the appropriate direction at the Creep Speed. If after the correction move, the difference between the Command and Actual positions is still not within the window, then the controller will attempt another correction move with the current positional error (multiplied by the Correction gain). The controller will therefore repetitively close the error until it is less than the Window or if a Time Out occurs. The correction is only allowed for a time defined by the Time Out parameter (set by the **TO** command in milliseconds) to complete the position error correction. If the correction is not within the Window after this time has elapsed, then a move *Not Complete* condition will be declared. The correction move scaling is set by the **KP** command in percent and therefore can take a value between 1 and 100. It should be set to avoid over-correction e.g. with a remote encoder and backlash.

2.13 Datum Search Strategies

The PM600 position controller has built in functions to allow an incremental drive system to find a reference position known as a datum. Usually the datum is found immediately after the controller has been powered up.
Once a datum has been established all subsequent moves are referenced to this position.
The PM600 captures the *actual* position in all closed-loop modes and the *command* position in open-loop stepper mode.
A datum can be the single pulse marker (index) on an encoder, a precision sensor or a simple switch. The choice of datum sensor is related to the accuracy and repeatability of the datum.
A *home to datum* move is initiated in the PM600 by a **HD** (**H**ome to **D**atum) command.

2.13.1 Datuming Speed

The speed of the *home to datum* (HD) moves is selected by the PM600's *datum approach* (DA) input.
For fast HD moves the DA input is linked to +VLL (or a voltage >10V). The speed during a fast HD move is set by the **SV** (**S**et **V**elocity) command. When a fast HD move is executed, the datum position is captured, the mechanism overshoots, slows down and then returns to the captured datum position.
For slow HD moves the DA input is left disconnected. The speed during a slow HD move is set by the **SC** (**S**et **C**reep speed) command. When a slow HD move is executed, the datum position is captured and the mechanism stops at the captured datum position.
A switch can be connected to the DA input to enable the mechanism to only slow down to the SC speed when approaching the datum. The mechanism decelerates as soon as the DA switch is opened. The DA switch is only required on systems where overshoot of the datum position cannot be tolerated or the *datum stop* (DS) sensor has a slow reaction time or a large hysterisis characteristic.

2.13.2 Using an Encoder Index Marker

The Index mark of an incremental encoder can be used as the datum for a mechanism. The Index mark channel of the encoder is connected to the *Encoder 1* input of the PM600 along with the A channel and the B channel. The PM600's *datum stop (DS)* input would be left disconnected.

2.13.3 Using a Precision Sensor

A +5V sensor such as a Sony Magnaswitch SET P15 (+5V) can be used to define the datum position. This is particularly useful in an open loop stepper system. The precision sensor is connected to the encoder index input. If the signal is differential then both the I+ and I- inputs are used. If the signal is single ended (e.g. TTL or open-collector) only the I+ input is used. The index mark input can be inverted using the DM command (see below).

2.13.4 Using a Switch

A switch can be used to define the datum position. The switch can be a standard microswitch, an optical switch a proximity switch or another 24V (>10V) device. The choice of measuring switch is dependent on the accuracy required. When using proximity switches a PNP type is required. The switch is connected from +VLL (or a voltage >10V) to the PM600's *datum stop (DS)* input. A proximity switch will require a 0V connection. The datum position is captured when the switch opens, so a normally closed type may be required depending on the mechanical system design.

2.13.5 Using a Switch and an Encoder Index Marker

In mechanisms where the encoder rotates a number of revolutions and only one particular index is required to define the datum position a switch can be used to mask the unwanted index markers. The switch is connected to the PM600's *datum stop (DS)* input and the Index mark channel of the encoder is connected to the *Encoder 1* input of the PM600. Both these signals are summed to create the final datum input. The datum position is therefore captured when both the switch opens and the next index mark occurs.

2.13.6 Datum Capture

As the datum position is captured when a datum event occurs, the HD command need not be used to capture the datum position. Just moving past the datum position will capture the current actual position as the datum position. Any previously captured datum position can be cleared using the **CD** (**C**lear **D**atum) command. A dot shows in the PM600's status display when a datum position is captured.

2.13.7 Automatic Direction Search

A datum sensor can be used to automatically define the direction that the mechanism moves to find the datum position (see below). The appropriate bit must be set using the DM command to select Automatic Direction search mode.

A datum sensor must be designed in to the mechanism such that over the range of movement it has a value of 0 (off) on one side of the datum position and 1 (on) on the other side of the datum position.

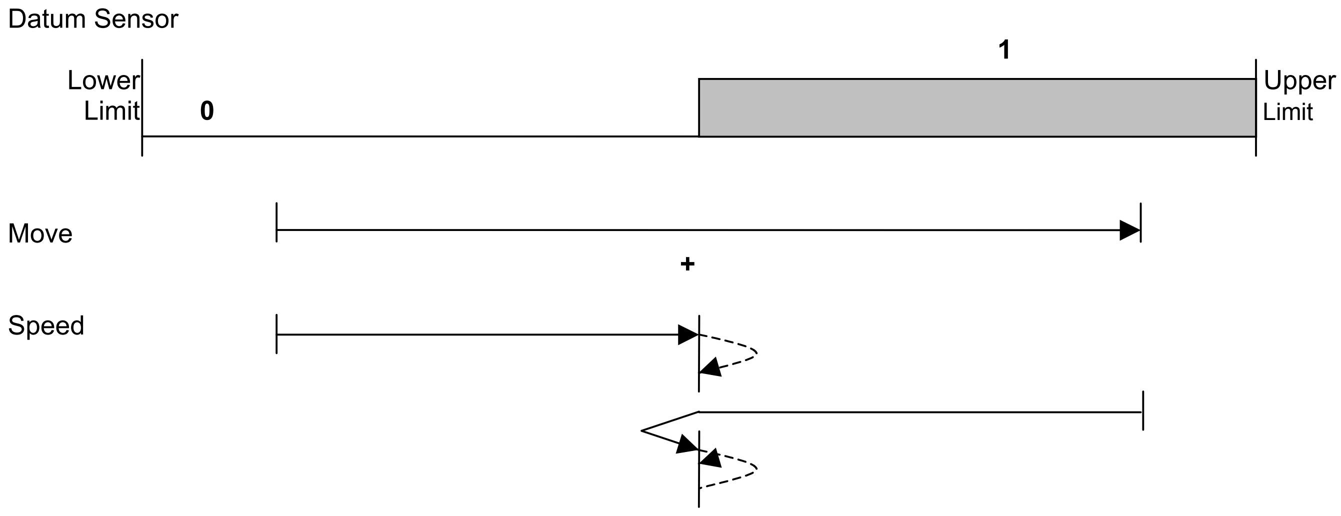
The datum sensor can be either an encoder index marker connected to the PM600's *Encoder 1* input or a precision sensor connected to the PM600's *datum stop (DS)* input.

When the HD command is executed the PM600 reads the value of the summed datum inputs to decide on which direction to move to try and find the transition from 0 to 1. If the value is 0 the HD move takes place in the positive direction. If the value is 1 the HD move takes place in the negative direction. The datum position will always be captured while moving in the same direction and the datum position capture always takes place on a 0 to 1 transition. If the HD move starts in the negative direction and the PM600 detects a 1 to 0 transition, it slows down and reverses to find the 0 to 1 transition. If the HD move starts in the positive direction and the PM600 detects a 1 to 0 transition it will capture the position.

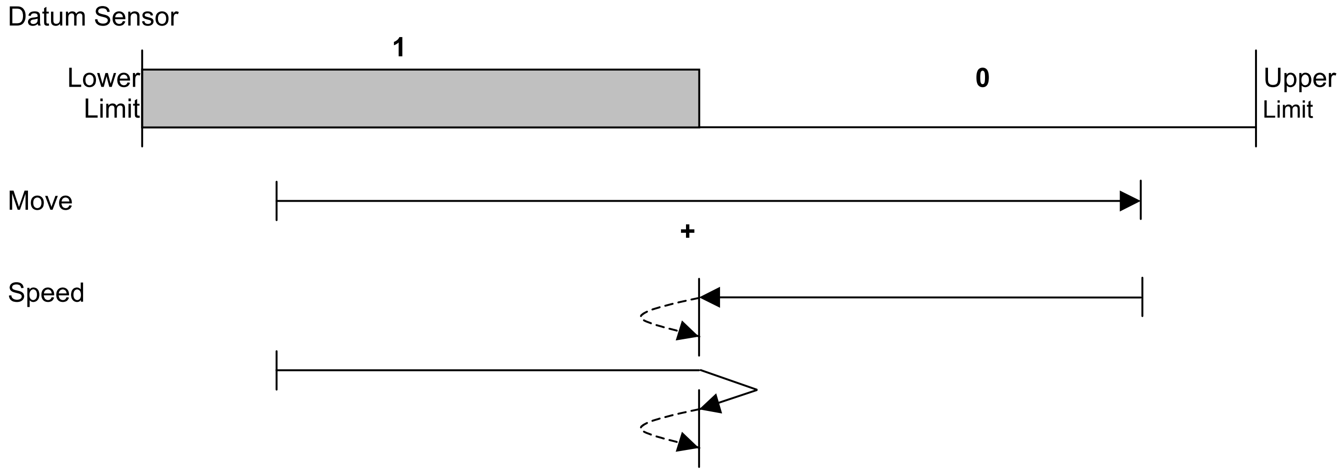
If the mechanism is moving too fast to stop at the datum position it will over shoot (indicated by the dashed lines) and return to the datum position. If the datum sensor has a value of 1 at the negative range of travel and a 0 at the positive end the command **HD-1** can be used to initiate a datum search in the opposite direction. The datum position is still captured on the 0 to 1 transition of datum input.

Note that the speed of the mechanism whilst searching for datum will be selected by the DA input (see above).

Command = HD



Command = HD-1



2.13.8 Automatic Opposite Direction Search

The PM600 can be configured to automatically move to a *Hard Limit* before starting its datum search. This is useful in applications where on power-up the direction to move to find the datum position is not known.
The appropriate bit must be set using the DM command to select Automatic Opposite Direction search mode.
Once this mode is selected any HD moves start by executing a constant velocity move towards the opposite Hard Limit. The speed of this section of the move is defined by the value set by the **SV** command. Care must be taken that the design of the Hard Limit switch mechanism can accommodate any overshoot caused by hitting a Hard Limit at speed. When the Hard Limit is reached a standard HD move is started.
A HD command will move to the lower Hard Limit and then search for a datum position in the positive direction.
A HD-1 command will move to the upper Hard Limit and then search for a datum position in the negative direction.

2.13.9 Datum Mode Command

The **DM** (**D**atum **M**ode) command is used to set the specific mode of operation of the HD command.

Important – if the set-up of the PM600 is changed (e.g. by using the DM or SH commands) then the **BD** (**B**ackup **D**igiloop) **must** be executed to save the set-up values to Flash memory. If this not done, the values will be lost on power-down.

The DM command has 8 bits associated with it, only 5 are defined at present. Each bit can be either 0 or 1 and selects the various following modes:

Bit **a** selects whether the Encoder index input polarity is inverted or not. This is only relevant to the Encoder Index input and not to the *datum stop* input. It is typically used when a normally-closed precision sensor is required to set the datum position. In this case the datum position is captured when the signal from the sensor goes from a high state to a low state (1 to 0).

Bit **b** selects whether the datum position is captured only once, typically after a HD command or each time the datum position passed.

Bit **c** can be used to force the datum capture event to set the datum position to the *Home Position* after a datum search using the HD command. The initial value of the *Home Position* is zero, but using the **SH** (**S**et **H**ome Position) command the value of the *Home Position* value can be changed to any more suitable value.

Bit **d** enables the Automatic direction search. (See above).

Bit **e** enables the Automatic opposite limit search (See above).

Bit **d** and bit **e** <u>can</u> both be set to 1 in which case the Opposite Limit search will be performed before the Automatic Direction search but this is not an appropriate method to use when moving to a datum position.

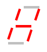
Bits **f**, **g** and **h** are for future use and should be set to zero.

2.13.10 Datum Search Diagnostics

Using the **OS** (**O**utput **S**tatus) command the status of the datum input can be found. This is the datum input after the encoder index marker and the DS input have been summed.
Bit 6 in the status string returned by the OS command shows the status of the datum input. Its value is 1 or 0 according to the following:

0	if DS input ≥ 10V **OR** if index = low
1	if DS < 10V or not connected **AND** if Index = high or not connected

Datum capture occurs when this signal goes from 0 to 1. It is edge triggered, not level triggered. A dot shows in the PM600's *status* display when a datum position is captured.

The STATUS display will show  during a Home to datum move.

2.14 Sequences

The PM600 can be programmed to execute a sequence of commands upon a prompt via the RS 232 data bus or from an external signal using a **WA** command.
Up to eight sequences (0 – 7) can be defined in the PM600.

Commands:

DS<n>	Start definition of sequence n.
ES	End definition of sequence.
XS<n>	Execute defined sequence n.
LS<n>	List sequence n definition.
US<n>	Undefine sequence n (clear sequence n from memory).
BS	Backup sequences to Flash memory.
AE<n>	Automatic execution of sequence n on power-up.
AD	Disable automatic execution.

Sequence Example:

Command	Function
1DS3	Define start of sequence 3.
1SV2000	Set slew speed.
1AP0	Define present position as zero.
1MA45000	First move (absolute).
1MR5000	Next move (relative).
1MR3000	Next move (relative).
1MA70000	Next move (absolute).
1SV200000	Set new slew speed.
1MA0	Next move (return to zero).
1ES	Define end of sequence.

The sequence can then be executed by the command: **1XS3**
If an **XS**<n> command forms part of the sequence, then control will jump to the beginning of the specified sequence. This method can be used to create continuous program loops. **XS** should be considered as a *goto* rather than a *gosub* command.
To escape from a sequence before completion the Control **C** or **ESC** should be used. This will stop a move, arrest the sequence and reset the controller to its idle state.

The sequence is defined in volatile memory and must be copied to the on-board Flash memory so that the sequence is retained after power-down. This is done using the *backup sequence* **BS** command.

The PM600 can be operated without the further presence of the host computer (i.e. *stand-alone*). The **AE**<n> command should be issued to instruct the controller to execute the programmed sequence n immediately on power-up. Issuing an AD command disables the automatic sequence execution.

For example:

1AE3	auto-execute sequence no. 3

The controller can be signalled to execute its sequence on receipt of an external signal. Use can be made of the *wait for input event* function **WA**. This command would be placed at the start of the program.

If the controller is waiting for signal/condition to become true a || is displayed on the STATUS display.

The *if true* **IT**, *if false* **IF** and **WA** commands can be used to create *smart* sequences.

Example 1:
This following sequence has been constructed to repeat a loop of moving in 400 step intervals, until read port 4 goes high (possibly activated by a switch).

1DS3	Start sequence definition
1MR400	Move 400 steps
1IF22221222	This condition is FALSE so next command is executed (i.e. NOT skipped).
1XS3	Condition was FALSE; therefore execute this sequence i.e. repeat this loop
(**1XS0)**	Return to main or another sequence (optional).
1ES	End sequence

The sequence starts by moving 400 steps. The **IF** command will then compare with the read ports, in this case it is only bit 4 that is relevant. If the condition is FALSE (the switch is not on), then the next command is executed which will restart the current sequence of move 400 steps and compare. If the condition becomes TRUE (the switch goes on), then the **XS3** command will be skipped and go on to the one after. This could be the end of the sequence or a command to then do another sequence such as returning to a master sequence.

Example 2:
If the following states are present on the inputs:

PORT :	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
STATE :	High	Low	Low	High	High	Low	Low	Low

1IT22222200	This condition is TRUE so next command is executed (i.e. NOT skipped).
1MR200	Move 200 steps
1IT22222201	This condition is FALSE so next command is skipped (i.e. is NOT executed).
1MR400	This command is skipped

If the following states are present on the inputs:

PORT :	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
STATE :	High	Low	Low	High	High	Low	Low	High

1IT22222200	This condition is FALSE so next command is skipped (i.e. is NOT executed).
1MR200	This command is skipped
1IT22222201	This condition is TRUE so next command is executed (i.e. NOT skipped).
1MR400	Move 400 steps

I.E. In the above example, read port 1 is used to select a move length and read port 2 will disable the move:

PORT :	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	
STATE :	(Ignored)	(Ignored)	(Ignored)	(Ignored)	(Ignored)	(Ignored)	Low	Low	Move 200 steps
STATE :	(Ignored)	(Ignored)	(Ignored)	(Ignored)	(Ignored)	(Ignored)	Low	High	Move 400 steps

2.14.1 Multiple Sequences

Up to eight *sequences* can be defined, these are numbered 0 to 7. They are executed using the **XS<n>** command, including an argument for the *sequence* number.

For example:

1XS3 Axis 1 - Execute sequence no. 3

If an **XS<n>** command is contained within a *sequence*, commands will execute from the beginning of that *sequence* until an **ES** command is encountered.

Any one defined sequence can be set to execute automatically on power-up. The **AE** command can include an argument that is the number of the sequence to execute.

The sizes of the *sequences*, *profile* and *cams* are pre-defined at 127 commands each.

To erase unwanted sequences from memory the following command is used.

US<n> undefine sequence

Note that this command only removes the sequence from memory until the unit is turn off and on again. A **BS** command should be issued to remove erased sequences from Flash memory.

2.15 Profiling

The PM600 can be programmed to execute a series of **MR** (move relative) commands without stopping between each move. Each section of the profile is performed at a constant velocity, so the more points defined in the move, the smoother the motion. A constant time interval between the points is set as part of the execute command. The profile does not have to be reloaded to run it at a different speed.

Up to eight profiles (0 – 7) can be defined in the PM600.

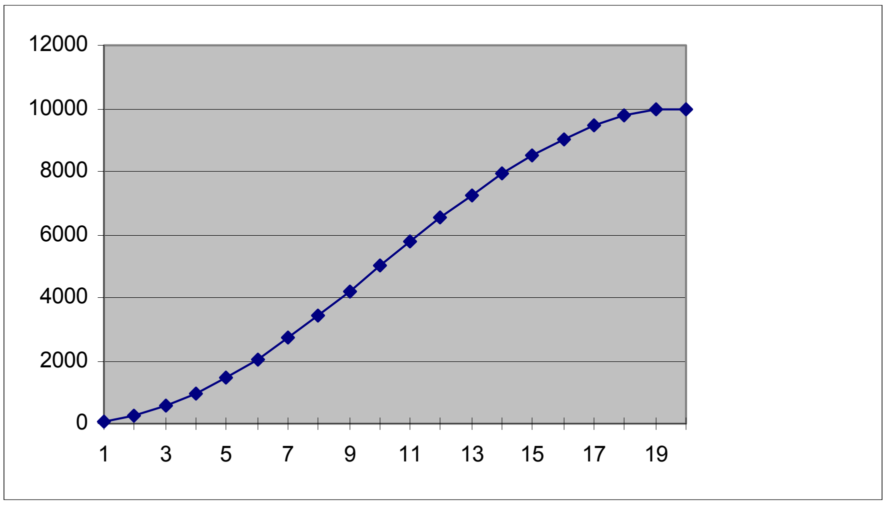
If required, a profile can be executed as part of a sequence. For optimum profiling response, the value of **KF** should be set equal the value of **KV**.

Commands:

DP<n>	Start definition of profile *n*.
EP	End definition of profile.
LP<n>	List definition of profile *n* (list profile moves).
MR<distance>	*Move relative* command to define the number of steps to be moved in successive time periods.
PT<time>	This command allows you to enter the time (in milliseconds) to complete each element in a profile definition.
XP<n>	Execute the defined Profile *n*. The move occurs at a rate, defined in milliseconds by the **PT** command, for each **MR** segment to be completed.
UP<n>	Undefine profile *n* (clear profile *n*).

Note that the **UP** command only removes the profile from memory until the unit is turn off and on again. A **BA** command should be issued to remove erased profiles from Flash memory.

Example:
The following profile was calculated using a cosine speed profile of a move of 10,000 steps split into 20 segments. During execution of the profile, each segment is completed in a time defined by the **PT** (profile time) command.



Profile Example

This profile would be defined using the following commands:

1DP1	Start profile 1 definition.
1MR61	Move relative no. 1.
1MR183	Move relative no. 2.
1MR300	Move relative no. 3.
1MR410	Move relative no. 4.
1MR510	Move relative no. 5.
1MR597	Move relative no. 6.
1MR669	Move relative no. 7.
1MR724	Move relative no. 8.
1MR763	Move relative no. 9.
1MR783	Move relative no. 10.

1MR782	Move relative no. 11.
1MR763	Move relative no. 12.
1MR724	Move relative no. 13.
1MR669	Move relative no. 14.
1MR597	Move relative no. 15.
1MR510	Move relative no. 16.
1MR410	Move relative no. 17.
1MR300	Move relative no. 18.
1MR183	Move relative no. 19.
1MR61	Move relative no. 20.
1EP	End profile definition.

The profile move is executed by the command **1XP1**. If the profile time was set to 250 ms (by the command **1PT250**) the move will take place in 5 seconds (20 x 250 = 5000 ms).

2.16 Electronic Gearbox

The PM600 controller can be used to slave the speed of its motor to that of another quadrature signal, usually from a *master* encoder (Input Encoder) mounted on another motor. When not in *gearbox* mode, the controller will still read the position of the input encoder.
The ratio of slaved velocity to that of the input encoder is variable *on the fly*.
The electronic gearbox mode is selected by sending a **GB** command. The controller will then immediately begin slaving its motor to the signal from the input encoder at a ratio set by the **GR** command.

Using the **GB** command the controller will enter gearbox mode and move relative to the position of the input encoder. If the **GA** command is used (absolute gearbox mode) the motor will only move when the input encoder is equal to the position (motor) encoder. This facility makes implementation of an electronic clutch possible.

The method for inputting the gearbox ratio is **GR**.

<address>**GR**<numerator>**/**<denominator>

For example:

2GR22/7	Set gearbox ratio to 22:7. For every 7 steps of the input encoder the motor will move 22 steps.

To allow the synchronisation position to be varied the commands **DA** (difference actual position) and **DI** (difference input position) are available to offset the positions.

Most applications require the slave motor to be *geared-down* with respect to the master. However, if the application requires a large *gear-up* ratio care should be taken in the selection of line counts on both encoders. If master and slave encoders have the same line count a small gear up ratio of 5:1 is normally OK. Very high ratios of 1000:1 and above leads to problems since the transition of a single encoder line on the master produces a large stepped error at the slave motor. For applications requiring a large *gear-up* ratio it is recommended that the slave encoder has fewer lines than the master. This could be achieved by scaling the positional encoder with the **ER** command.

2.16.1 Gearbox Commands

Whilst in gearbox mode, the PM600 will queue (not act upon) **MA**, **MR** and **CV** commands. All other dynamically-related commands will be accepted.

GR Gearbox ratio. In gearbox modes the ratio is specified by two arguments separated by a **/** character. The ratio is therefore specified as a fraction with the format: numerator**/**denominator. This ratio is also used for input encoder jog scaling. The **GR** command cannot be used in a sequence, instead use **GN** or **GD** to change ratios in sequences.

GN Set gearbox ratio numerator. This command can be used in conjunction with the **GD** gearbox denominator. The ratio is therefore specified as a fraction with the format: numerator(**GN**)/(**GD**)denominator. The range of the numerator value is –32768 to 32767 ($\pm 2^{15}$).

GD Set gearbox denominator. The range of the denominator is 1 to 32767 (2^{15}).

Example:

1GN2	
1GD5	Axis 1 Set electronic gearbox ratio to 2:5 - i.e. for every 5 steps of the input encoder the command position will change by 2 steps.

GB Enter gearbox mode. The response to a **CO** (Current Operation) command while in gearbox mode is **01:Gearbox**.
Example: 1GR1/4
1GB
In this case, the PM600 will drive the slave motor at a rate 1/4 of the speed of the master encoder.

GA Enter gearbox mode when the value of the input encoder is equal to the value of the position encoder (absolute gearbox mode). Operation is then the same as above. The response to a **CO** (Current Operation) command while the *input position* is moving towards the *actual position* is **01:Gearbox synchronisation**.

GM When in gearbox mode, the **GM** (Gearbox Move Relative) command can be used to superimpose a relative move on top of the gearbox slaving. In this way, a correction in the synchronism of the two positions can be changed without exiting the gearbox mode. This move is done at the creep speed. The response to a **CO** (Current Operation) command while this move is taking place is still **01:Gearbox**.
Example:
1GM100 Superimpose a move of 100 steps (positive) on top of gearbox mode.

ST Stops motor and return to normal mode.

To reverse the direction of the motor with respect to the input encoder a negative value of gear ratio should be used.
Example: 1GR-1/4

2.17 Software Cam

The Cam Profile function allows you to control the position of a servo, in relation to the position of a master encoder (or other axis of motion). This master position is fed to the PM600 using Encoder Input 3 and is shown internally as the Input Position or IP.

The start and finish commands for defining a cam are **DC** and **EC**. Up to eight CAM profiles can be defined and stored in the PM600 (0 - 7). Cams are executed using **XC** or **XR**. Cam co-ordinates are input using the **XY** command:

<address>**XY**<x co-ord.>**/**<y co-ord.>

Cam Commands:

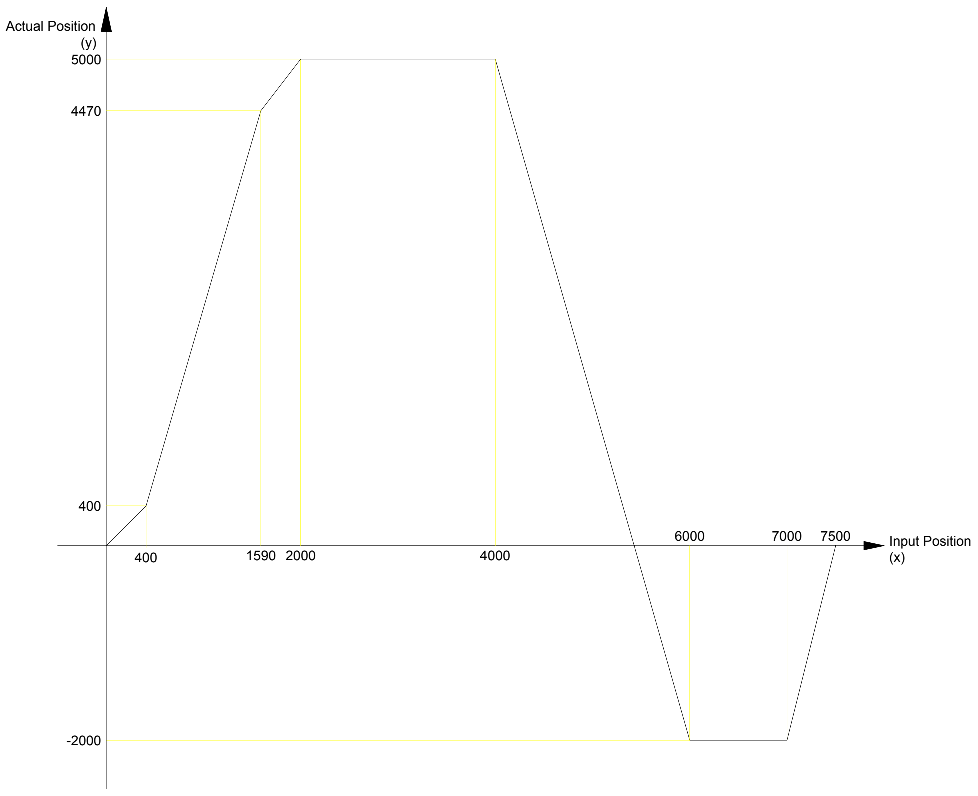
DC<n>	start cam *n* definition.
EC	end cam definition.
LC<n>	List cam definition *n* (list cam points).
UC<n>	undefine cam *n* (clear cam from memory).
XC<n>	execute cam *n*

CO returns **Cam Synchronising** or **Cam**

2.17.1 Cam Definition:

Cam profiles are *piecewise linear*, with the first co-ordinate implicitly (x=0, y=0). Co-ordinate pairs must be defined in order of increasing x co-ordinate. There can be up to 128 points defined for each profile.
The x co-ordinate can be in the range 0 to 2147483647 (2^{32}) and the y co-ordinate can be in the range –2147483647 to 2147483647($\pm 2^{32}$). The x co-ordinate of the last pair defines the *modulo*, that is the repeat distance. In the example given below, the modulo is 7500, so that the y values for x=2000, x=9500, x=17000, etc. are the same.

Example:



CAM Example

The above cam profile would be defined (into CAM 2) using the following commands:

```
1DC2
1XY400/400
1XY1590/4470
1XY2000/5000
1XY4000/5000
1XY6000/-2000
1XY7000/-2000
1XY7500/0
1EC
```

You can then view the programmed cam with the List Cam command (1LC2).
You can remove a cam definition with the Undefine Cam command (1UC2).

A valid cam definition command will receive a response
OK: STORED IN CAM
An invalid cam definition such as a repeated position will receive a response
! NOT MONOTONIC

The currently defined cam profiles can be stored into the non-volatile, flash memory with the Backup Cams command (**BC**). When the controller is subsequently powered up, it will have these cams set.

2.17.2 Cam Operation:

The cam profile is either started by using the execute cam command **XC** (absolute cam) or the execute cam relative command **XR**.

The above cam would be started either by the command **1XC2** or the command **1XR2**.

Whilst operating in cam mode the STATUS display will show C .

Using the **XC** command the motor will only start to move when the *input position* (Encoder 3) divided by the *cam modulo* is equal to the equivalent *actual position* (Encoder 1) defined in the *cam* profile. The response to a **CO** command while the PM600 is waiting for the positions to synchronise is **01:Cam synchronisation**.

Important when using an absolute cam, the actual position must be within the range specified in the cam definition for synchronisation to be achieved.

Using the **XR** command the motor will start to move immediately there is a change in the *input position.* The response to a **CO** command while the PM600 is executing a *cam* using either the **XC** or **XR** commands is **01:Cam**

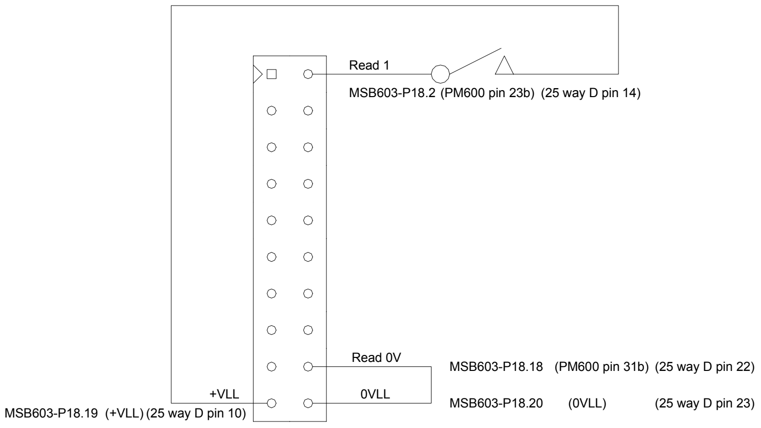
While in *cam mode* enquiry commands such as **QA** (query all), **OA** (output actual position), etc. can still be performed. Exit from *cam mode* can be achieved by either **ESC** (escape) or **ST** (soft stop) commands.

To obtain the most accurate cam action the feedforward coefficient should be made equal to the velocity coefficient. **KF**=**KV**.

2.18 Interfacing To Analogue And Digital I/O

2.18.1 Digital Inputs

There are eight digital inputs that can be used to interface with the PM600. These inputs are known as *read ports* 1 to 8. These inputs are opto-coupled with a common 0V for all *read ports*. An input voltage of between +10V and +35V (1) activates them. If a *read port* is unconnected it will read as **0**.


Read 1
MSB603-P18.2 (PM600 pin 23b) (25 way D pin 14)
Read 0V
MSB603-P18.18 (PM600 pin 31b) (25 way D pin 22)
+VLL
0VLL
MSB603-P18.20 (0VLL) (25 way D pin 23)
MSB603-P18.19 (+VLL) (25 way D pin 10)

The input can be connected to either a PNP signal output, a switch, or another controller's output *write port*.

The **RP** Read input Port command is used to check the operation of the *read ports*. This instruction returns an eight digit binary number of either **0** or **1** characters to represent the current state of the read port. These start with read port 8, through to 1. Referring to the diagram above, a **1** represents a closed switch and a **0** represents an open switch.

The PM600 can be programmed to execute a sequence of commands upon a prompt from an external signal using a **WA** command.
This command will examine the read port inputs and compare them with the specified bit pattern argument. It will wait until the inputs are equal to the specified bit pattern before issuing its **'OK'** response and moving on to the next command.
The bit pattern is specified as an eight digit binary number of either **0, 1** or **2** characters starting with read port 8, through to 1. A **1** defines that the input must be high (+24V), a **0** defines that the input must be low (0V or open-circuit) and a **2** defines that the input is not relevant or 'don't care'. If less than eight digits are specified in the argument, then the preceding ones are assumed as low (**0**).

Example:

1WA22222210	Wait until the following condition is on the read input port before continuing:							
PORT:	8	7	6	5	4	3	2	1
STATE:	(Ignored)	(Ignored)	(Ignored)	(Ignored)	(Ignored)	(Ignored)	High	Low

Sequence Example:

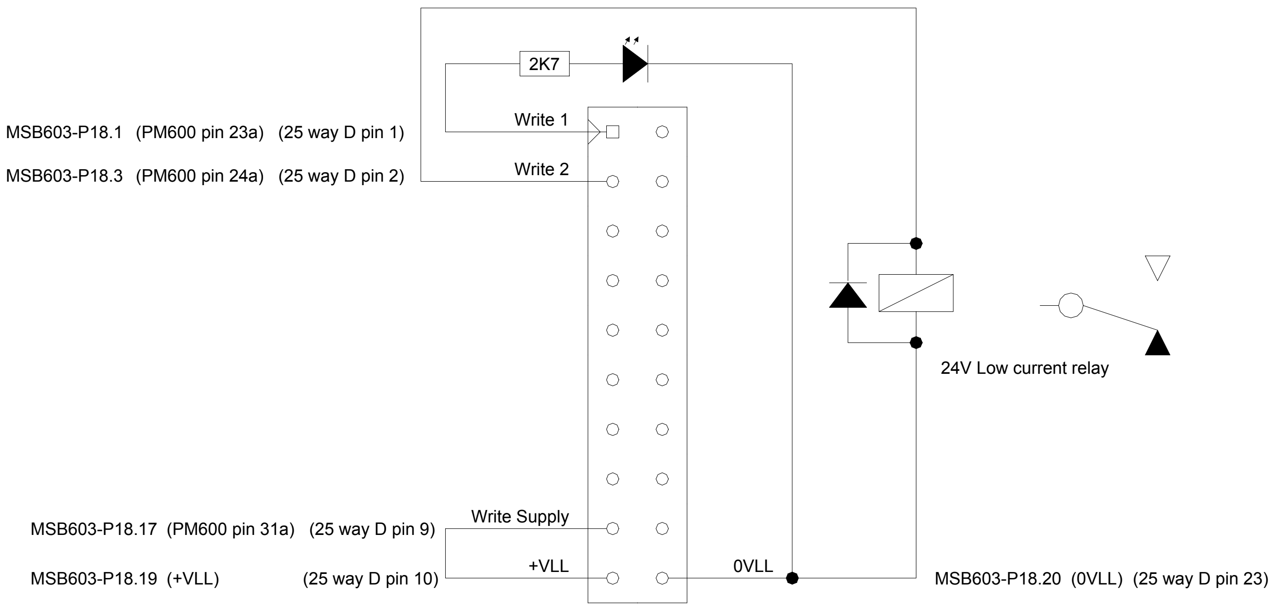
The following sequence can be used to allow a motor to be indexed when a digital input is turned on. A digital output is switched on when the move is complete. The sequence then waits for the digital input to be turned off before repeating the sequence.

Command	Function
1DS2	Define start of sequence 2.
1WA22222221	Wait for Read Port 1 (digital input 1) to be turned on.
1WP0	Turn all Write Ports (digital outputs) off.
1MR2000	Move relative 2000 steps (index).
1WE	Wait for end of move
1WP22222221	Turn Write Port 1 (digital output1) on.
1WA22222220	Wait for Read Port 1 (digital input 1) to be turned off.
1XS2	Repeat sequence 2 (loop).
1ES	End of sequence.

The sequence can then be executed by the command: **1XS2**

2.18.2 Digital Outputs

The PM600 controller has eight output ports, known as *write ports* 1 to 8. These outputs are driven by opto-couplers with a common V_{source} (write supply) connection. The outputs can be connected to an indicator (LED) opto-isolator, a low current relay or another controller's input *read port*.
Important when driving an inductive load such as a relay, the outputs must be protected by an external diode. Do not reverse bias these outputs.


2K7
Write 1
Write 2
MSB603-P18.1 (PM600 pin 23a) (25 way D pin 1)
MSB603-P18.3 (PM600 pin 24a) (25 way D pin 2)
24V Low current relay
Write Supply
MSB603-P18.17 (PM600 pin 31a) (25 way D pin 9)
+VLL
0VLL
MSB603-P18.19 (+VLL) (25 way D pin 10)
MSB603-P18.20 (0VLL) (25 way D pin 23)

The **WP(bit pattern)** command is used to write to the output port. The bit pattern is specified as an eight digit binary number. The digits will be either characters **0, 1** or **2** starting with *write port* 8, through to 1. A **0** defines that the output will be **off**, a **1** defines that the output will be **on** and a **2** defines that the output will not change from its current state.

The power-on states of the *write ports* are **00000000** - i.e. all **off**. The results of the last write port (**WP**) command is displayed on the **QA** page.

2.18.3 Analogue Inputs

There are five analogue inputs. Inputs 1 and 2 are general purpose bipolar inputs and 3 to 5 are for use with Joysticks although they can be used as general purpose unipolar inputs. All analogue inputs are converted to digital values by a 12 bit analogue to digital converter (ADC).

Input	Use	Type	Voltage	Values	Scaling
1	General purpose	Bipolar	-10V to +10V	–2047 to +2047	~5mV per unit
2	General purpose	Bipolar	-10V to +10V	–2047 to +2047	~5mV per unit
3	Joystick 1	Unipolar	0V to 5V	0 to 4095	~1.2mV per unit
4	Joystick 2	Unipolar	0V to 5V	0 to 4095	~1.2mV per unit
5	Joystick Centre Tap	Unipolar	0V to 5V	0 to 4095	~1.2mV per unit

The command has the syntax **<ad>AIn** where <ad> is the axis address and **n** is the analogue input.
The response is a string of numeric characters representing the value. If the analogue input selected is not in the range 1 to 5 the error message **! NOT VALID ADC CHANNEL** will be received.
Example:

If the controller of axis 1 currently has a Joystick 1 signal of about 2.5V then the command:
1AI3 will respond: **01:2040**

To allow the analogue inputs to be used as conditional inputs the commands 'Wait for analogue input greater than' (**AG**) and 'Wait for analogue input less than' (**AL**) are available.

The commands have the syntax **<ad>AGn/xxx** and **<ad>AGn/xxx** where <ad> is the axis address, **n** is the input selected and **xxx** is the input value that causes the condition to become true.
The response is either **OK** if the condition has been accepted and come true or if the analogue input selected is not in the range 1 to 5 the error message **! NOT VALID ADC CHANNEL** will be received.
Examples:

1AG2/1000	Wait until Analogue Input 2 is greater than 5V
1AG3/2047	Wait until Joystick 1 Input is greater than 2.5V
1AL2/-500	Wait until Analogue Input 2 is less than 2.5V
1AL3/2047	Wait until Joystick 1 Input is less than 2.5V

2.18.4 Analogue Outputs

There are two analogue inputs. Analogue output 1 is used by the PM600 in servo mode to drive a servo amplifier, although it can be used as general purpose output in stepper control modes. Both analogue outputs are bipolar and are converted from digital values by a 12 bit digital to analogue converter (DAC).

Output	Use	Type	Voltage	Values	Scaling
1	Servo/General purpose	Bipolar	-10V to +10V	–2047 to +2047	~5mV per unit
2	General purpose	Bipolar	-10V to +10V	–2047 to +2047	~5mV per unit

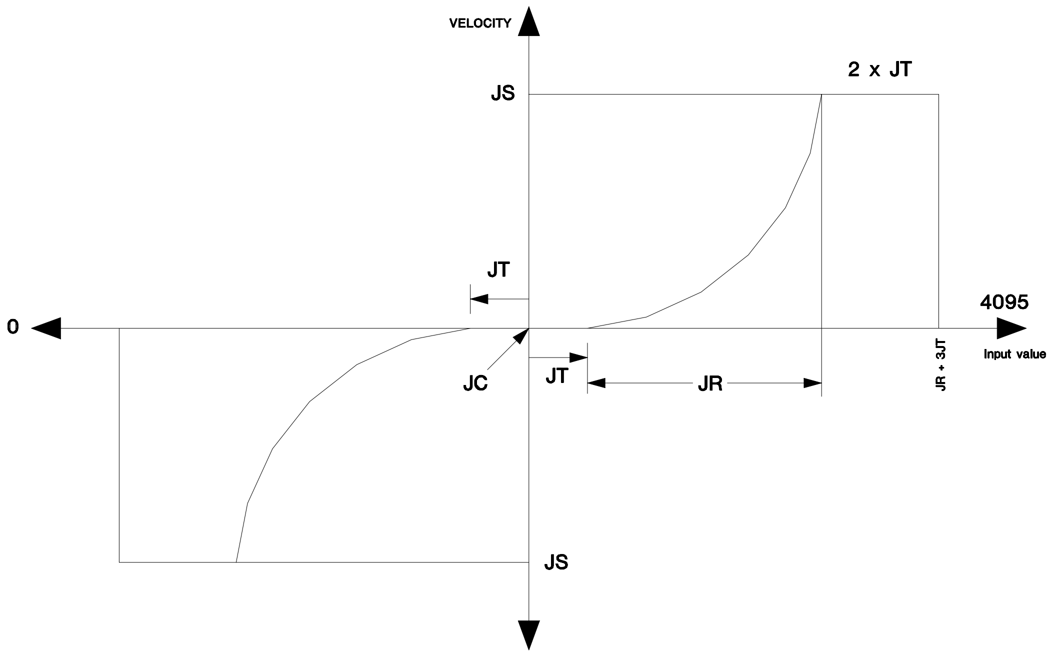
The command has the syntax **<ad>AOn/xxx** where <ad> is the axis address, n is the output selected and xxx is the output value.

The response is either **OK** if the command has been accepted or if the analogue output selected is not 1 or 2 the error message **! NOT VALID DAC CHANNEL** will be received.

Examples:

1AO1/1024	Set the Analogue output 1 to +5V
1AO2/-2047	Set the Analogue output 2 to -10V

2.19 Joystick Calibration



IF JC=0 it uses Joystick - CT input

Joystick Calibration

The joystick is powered by the +5V supply from the PM600.

There are various commands that are used to calibrate the operation of the joystick control.

AIn	Read the value of analogue input n.
JC	Joystick centre – used to set the zero point of the joystick.
JR	Joystick range – used to set the range of the joystick.
JS	Joystick speed – used to set the maximum speed when using joystick control.
JT	Joystick Threshold – used to set the amount of joystick movement before any move occurs.
JM	This command selects the input or inputs that are used for jogging.
QJ	Query the current Joystick settings.

The Joystick signal is connected to Analogue Input 3 and the Joystick Centre Tap is connected to Analogue Input 5. The values of these signals can be found using the **AI** command.

The procedure is:

1. If the Joystick does not have a centre tap connection then set the value of the Joystick centre. Use the **AI3** command to find the value when the joystick is at rest and use the **JC** command to set it.

2. Set the Joystick Threshold using the **JT** command. Once again the **AI3** command can be used to find a suitable value. Too low a value will cause the controller to start Jogging, too high a value will give an excessive dead band of operation. If the Joystick jogging mode is enabled using a **JM0100000** command, by looking at the front panel status display, the mode of operation can be seen.

   A on the display indicates a *Joystick* move whereas a indicates *Idle.*

3. The range value set by **JR** is calculated. Before the calculation is done the full range value from the Joystick must be found. Moving the joystick to its full range and then reading the value using an AI3 command does this. If the value of **JC** is set to 0, the value of the zero position (centre tap) input can be found using the **AI5** command.

$$\boxed{\text{The equation is <range> = <full range> - <2 x threshold> - <zero position>.}}$$

4. The maximum move speed is set by the **JS** command in steps per second.

The current Joystick values can be found using the Query Joystick (**QJ**) command. To save the values to FLASH memory use the **BD** command.

Example:

A Joystick is connected to the +5V power supply, 0V, the Joystick Input and the Joystick Centre Tap input.
The Joystick centre value is found to be 2051 using **AI5**.
The Joystick threshold is 25 found empirically by moving the Joystick slightly from its zero position and using the **AI3** command to find the value then calculating the difference between this number and the centre value (2076 – 2051 or 2051 - 2026) = 25.
The Joystick full range is found to be 2460 by moving the Joystick to its furthest positive extent and using the **AI3** command. The Joystick range is then calculated to be 2460 – (2 x 25) – 2051 = 359.
The maximum required Joystick speed is 8000 steps/sec.

The commands to set these values on axis 1 are:

1JC0	Use Joystick centre tap for centre value.
1JT25	Set Joystick Threshold to 25.
1JR359	Set Joystick Range to 359
1JS8000	Set maximum Joystick speed to 8000 steps/sec.

2.20 Communication

The PM600's DIP Switch, SW3 is used to set the serial communication parameters. These should be set to match those of your host terminal or PC, and all PM600 units should be set the same. If the unit receives characters that do not match the set parameters it will cause a communication error abort. A Γ will be shown on the front panel STATUS display if there is a communication error. This is most likely to be caused by a mismatch between the PM600 setup and the controlling device's setup.

The PM600 commands are received by the RS232 port as strings of ASCII characters. There is no protocol associated with the strings so a simple terminal program can be used to communicate with the PM600.
The type of interface used can either be RS232 or RS485. This is selected on the PM600 by SW3-7.
Interface type used

7 ■□	7 ■□ RS232	7 □■ RS485

2.20.1 RS232 Interface

The RS232 interface is configured either in a three-wire format or a five-wire format using Clear to Send (CTS) and Request to Send (RTS) handshaking. SW3-4 sets the type of interface used by enabling or disabling CTS and RTS.

Hardware handshake (RTS / CTS)

4 ■□	4 ■□ Disabled	4 □■ Enabled

The characters transmitted from the controlling device are echoed back to the terminal along with the replies. On multi-axis systems a daisy-chain configuration is used whereby the transmit output (Tx) of one controller is connected to the receive input (Rx) of the next controller. The Secondary comms connector is used for connecting either to the next axis or to an RS232 loop-back terminator. The RS232 Loop-Back Terminator simply connects Rx and Tx to allow both replies from the controllers and commands sent to the controllers to be echoed back to the controlling device.

2.20.2 RS485 Interface

The RS485 interface bus can be used in a multi-drop configuration. To avoid data collisions each reply must be read before the next command is sent.
A 100Ω terminator should be fitted to the end of the RS485 bus. This can be done by turning on the terminator on the last PM600 on the bus using SW3-8.
RS485 terminator select

8 ■□	8 ■□ No terminator	8 □■ 100Ω terminator

The PM600 has no biasing resistors fitted on board, so they must be fitted externally.

Note that in RS485 mode, received characters are not echoed back.

2.20.3 Quiet and Verbose Modes

The PM600 can be configured in *Quiet* or *Verbose* modes using SW3-5.
Reply format

5 ■□ 5 ■□ Verbose mode 5 □■ Quiet mode

The default setting is *Quiet.* Many responses are a string of numeric characters. If the communications are in *Verbose* mode, the reply is preceded by a string of characters such as **Following error =**
In *quiet* mode every command with the exception of **Esc** and **Ctrl C** results in a reply. In *verbose* mode messages do not have to be preceded by a command. Error messages will be transmitted from the PM600 spontaneously.
In *verbose* mode various messages can be transmitted from the PM600 on power up.
Example
01:Restoring parameters
01:Mclennan Digiloop Motor Controller V5.16a(0.6): Multi-stepper mode (0.1)
01:Restoring sequences -> Valid sequences: 0 (Autoexec 0 enabled)
01:Restoring profiles -> Valid profiles: none
01:Restoring cams -> Valid cams: none
If the PM600 is programmed to run *sequence* 0 on power up so the following message is also transmitted.
01:Autoexecuting sequence 0.

Example

Verbose Mode -	Responses are given in full (e.g. **1OS** gives 01:Status = 00000000). Unsolicited responses are allowed (e.g. **01:! STALL ABORT**).
Quiet Mode -	Responses are given abridged (e.g. **1OS** gives 01:00000000). Unsolicited responses are NOT allowed (e.g. **01:! STALL ABORT**).

Verbose mode is useful for human communication and quiet mode should be used for computer sent commands.

2.21 Command Word Syntax

The PM600 has a wide range of command options extending beyond the main move functions. The aim is to provide a flexible and comprehensive control device for integration of motion control into larger systems.

2.21.1 Commands

Most commands are two letters, the function of each, being described in section 7. Each command is preceded by the appropriate address to identify the axis for which the command is intended.
Where applicable (e.g. move commands, setting of system parameters, etc.) the command should then be followed by the desired value:

aXXnnn<cr>

a = address
XX = command
nnn = value (if required)
<cr> = carriage return.

Command strings should be terminated with a carriage return character (ASCII 0D hex).
Upper or lower case characters can be used for the command. Spaces within the command line are ignored.
All commands except for Control C and ESC are buffered. Commands are executed in consecutive order. Commands will be acted on sequentially, as they have been entered. If any command cannot be executed immediately (because it may need to wait for some condition or a previous command to finish) then the command and any that follow it will be buffered internally (up to 256 characters).

NOTE: the controller does NOT detect Delete, backspace and cursor movement characters. With some terminals or emulators these keystrokes will be translated as an escape sequence, i.e. a sequence of characters beginning with an escape character (ASCII 1B hex). The controller will detect the escape character and act on it accordingly.

2.21.2 Replies

Response to a command, once it has been accepted, is either an **OK** string or an alpha-numeric string terminated in a carriage return character (0D) and a line feed character (0A). An appropriate message is sent if either a mistake or conflicting instruction creates an error. The first character of an error message after the address prefix is **!** (the fourth character).

A reply is prefixed with the axis address and a colon, i.e. **01:OK** (useful in a multi-axis system).

2.22 Privilege Levels

The PM600 has a method of restricting the commands available to users. Each command has a 'Privilege Level' allocated to it. The 'Privilege Level' is set using the **PL** command, the higher the privilege level the more commands that are available. Before changing the 'Privilege Level' a security code known as a personal identification number (PIN) must be given. This is done by the **PI** command. Once the PIN has been entered the privilege level can be changed. If the privilege level is changed then a backup Digiloop (**BD**) command must be issued to store the new privilege level to flash memory.

The privilege level for any command can be found either on the PM600's help pages or in the programmers reference section of this manual. The commands are allocated to the following privilege levels.

0	Help and Query commands.
1	Moves, Wait and Write port commands.
2	Execute Sequences.
3	Set or change positions.
4	Set Joystick parameters and save parameter data to Flash memory.
5	Define and save sequences.
6	Set parameters and modes.
7	Set servo parameters.
8	Initialise values and change control mode.
9	Privilege Level or Pin change.

The commands associated with the privilege level system are:

NP<new PIN>	***N***ew ***P***in
PI<pin>	Enter ***PI***N
PL<level>	Set ***P***rivilege ***L***evel
QL	***Q***uery Privilege ***L***evel

Note that once a valid PIN has been entered full access is given to all commands irrespective of the currently set privilege level. In this case a **QL** command will report the privilege level as *full* followed by the set value in brackets. E.g. **01:Privilege level = full (8)**. Full access will be cancelled either by entering an invalid PIN or by turning off the power to the PM600.

3 Installation

3.1 Physical Installation

The controller is constructed on a single EXTENDED EUROCARD standard printed circuit board. The dimensions of the PCB are 100mm x 220mm. It is designed for mounting in a 3U high 19" rack and is fitted with front panel that is 7HP (35.2mm) wide.
Connections are made via a 96 pin a, b & c DIN41612 type C connector. A mating half connector can be fitted in the 19" rack or preferably use a Mclennan PCB motherboard. The MSB603 motherboard has been specifically designed for installation of the PM600 controller. It has the DIN41612 socket on one side and plug-in screw terminals, IDC connectors and Molex terminals on the other for external connections.

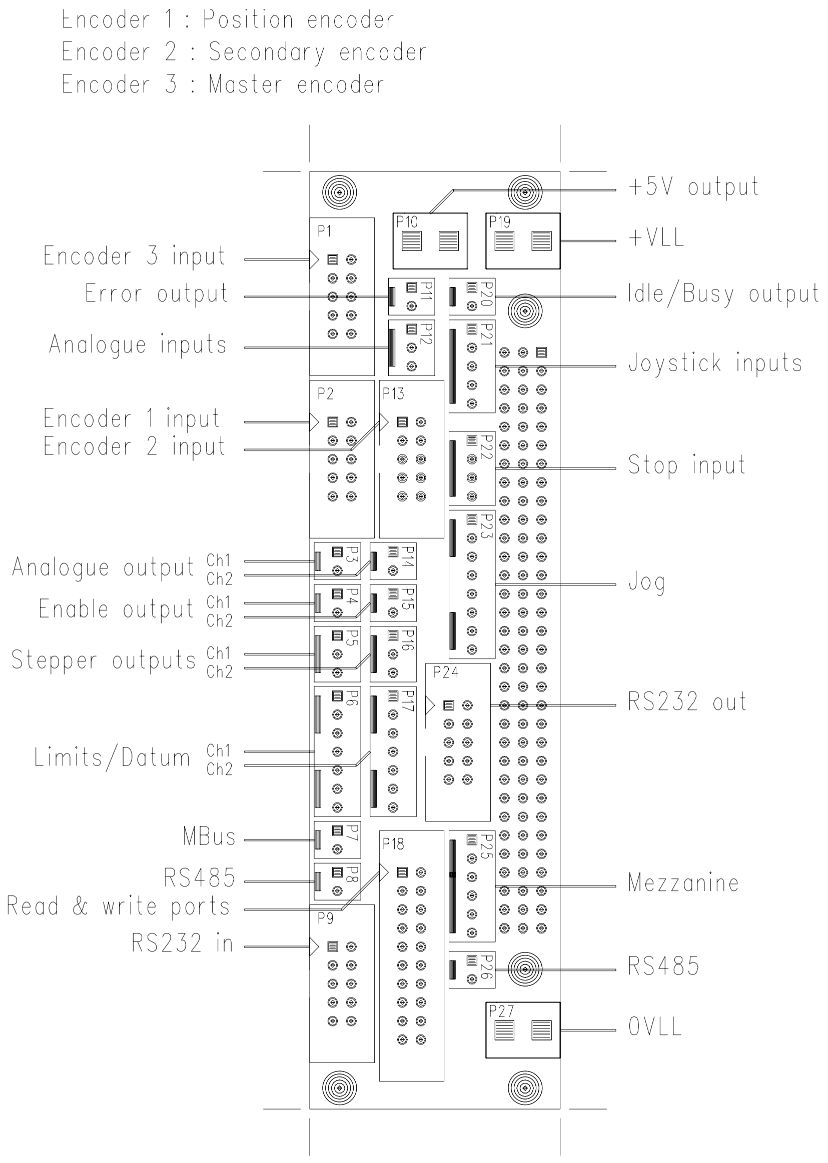
3.2 External connections

Electrical connections to the PM600 controller are made via the standard 96 way DIN41612 connector.
The pin assignments are as follows:

	A	B	C	
01	+VLL supply	+VLL supply	+VLL supply	01
02	Idle Output V_{source}	Error Output V_{source}	Analogue Input 1	02
03	Idle Output	Error Output	Analogue Input 2	03
04	Joystick 1 (Analogue In 3)	Joystick 2 (Analogue In 4)	Joystick Centre (Analogue In 5)	04
05	+5V output	+5V output	+5V output	05
06	Encoder 1 A+	Encoder 2 A+	Encoder 3 A+	06
07	Encoder 1 A-	Encoder 2 A-	Encoder 3 A-	07
08	Encoder 1 B+	Encoder 2 B+	Encoder 3 B+	08
09	Encoder 1 B-	Encoder 2 B-	Encoder 3 B-	09
10	Encoder 1 I+	Encoder 2 I+	Abort Stop Input	10
11	Encoder 1 I-	Encoder 2 I-	Abort Stop Isolated 0V	11
12	Analogue Output 1	Analogue Output 2	Jog Select Input	12
13	Analogue 0V	Analogue 0V	Jog Positive Input	13
14	Channel 1 Enable V_{source}	Channel 2 Enable V_{source}	Jog Fast Input	14
15	Channel 1 Enable Output	Channel 2 Enable Output	Jog Negative Input	15
16	Channel 1 Step Output	Channel 2 Step Output	Jog Channel Output	16
17	Channel 1 Direction Output	Channel 2 Direction Output	Jog Isolated 0V	17
18	Channel 1 Upper Hard Limit	Channel 2 Upper Hard Limit	RS232 Transmit (Out)	18
19	Channel 1 Lower Hard Limit	Channel 2 Lower Hard Limit	RS232 Receive (In)	19
20	Channel 1 Datum Approach	Channel 2 Datum Approach	RS232 RTS (Out)	20
21	Channel 1 Datum Stop	Channel 2 Datum Stop	RS232 CTS (In)	21
22	Channel 1 Limit/Datum Iso 0V	Channel 2 Limit/Datum Iso 0V	RS485-A	22
23	Write Port 1	Read Port 1	RS485-B	23
24	Write Port 2	Read Port 2	M-bus Data	24
25	Write Port 3	Read Port 3	M-bus Clock	25
26	Write Port 4	Read Port 4	Mezzanine Option 1	26
27	Write Port 5	Read Port 5	Mezzanine Option 2	27
28	Write Port 6	Read Port 6	Mezzanine Option 3	28
29	Write Port 7	Read Port 7	Mezzanine Option 4	29
30	Write Port 8	Read Port 8	Mezzanine Option 5	30
31	Write Port Source	Read Port Isolated 0V	Mezzanine Option 6	31
32	0VLL ground	0VLL ground	0VLL ground	32

Connections can be made either directly to the edge connector or via the MSB603 motherboard described below.

3.3 MSB603 Motherboard

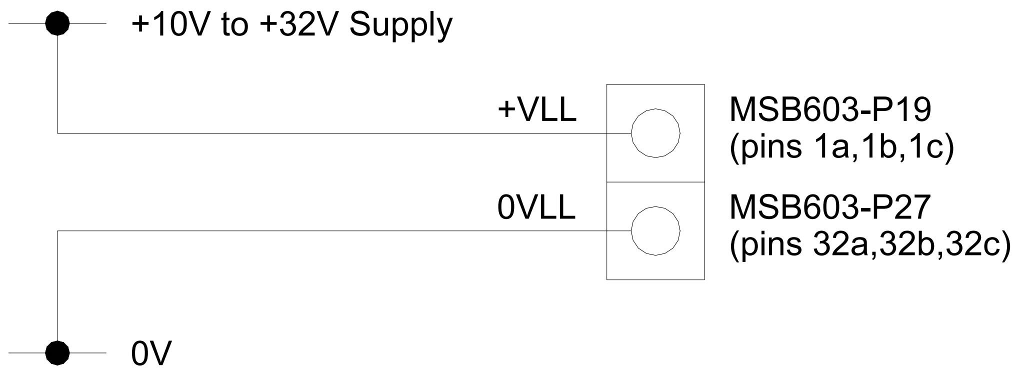


Ref.	Function	Type
P1	Encoder 3 Input	10 way IDC plug
P2	Encoder 1 Input	10 way IDC plug
P3, P14	Analogue Output	2 way ‘Molex’ plug
P4, P15	Enable Output	2 way ‘Molex’ plug
P5, P16	Stepper Outputs	3 way ‘Molex’ plug
P6, P17	Limits/Datum	7 way ‘Molex’ plug
P7	Mbus	2 way ‘Molex’ plug
P8, P26	RS485	2 way ‘Molex’ plug
P9	RS232 In	10 way IDC plug
P10	+5V Output	2 way screw-terminal plug
P11	Error Output	2 way ‘Molex’ plug
P12	Analogue Inputs	3 way ‘Molex’ plug
P13	Encoder 2 Input	10 way IDC plug
P18	Digital I/O	20 way IDC plug
P19	+VLL	2 way screw-terminal plug
P20	Busy/Idle Output	2 way ‘Molex’ plug
P21	Joystick Inputs	5 way ‘Molex’ plug
P22	Stop Input	4 way ‘Molex’ plug
P23	Jog	8 way ‘Molex’ plug
P24	RS232 Out	10 way IDC plug
P25	Mezzanine	6 way ‘Molex’ plug
P27	0VLL	2 way screw-terminal plug

3.4 Power Supply

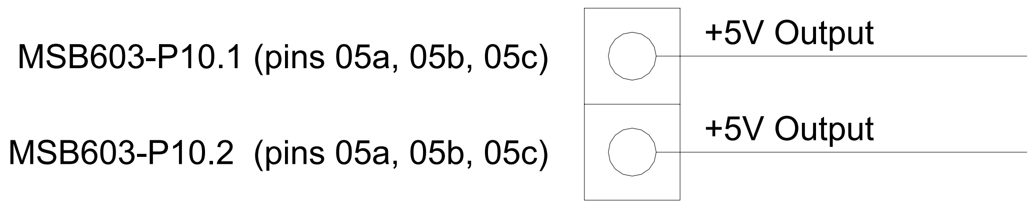
An on board switched-mode regulated power supply allows the PM600 to be supplied from a single unregulated DC source of between +10V and +32V. If the DC supply to the PM600 is switched then add a 10Ω 5W resistor in series with the supply.

THIS UNIT MUST NOT BE REVERSE POLARISED!



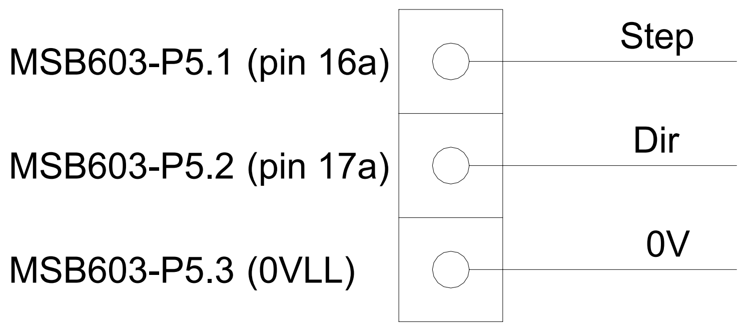
3.5 5V Output

The +5V supply that is generated by the PM600 on board switched-mode power supply is available to power external devices. If using an MSB603, this supply is taken to the encoder connectors and screw terminals. The total current available from the +5V supply is 500mA and care must be taken not to exceed this.

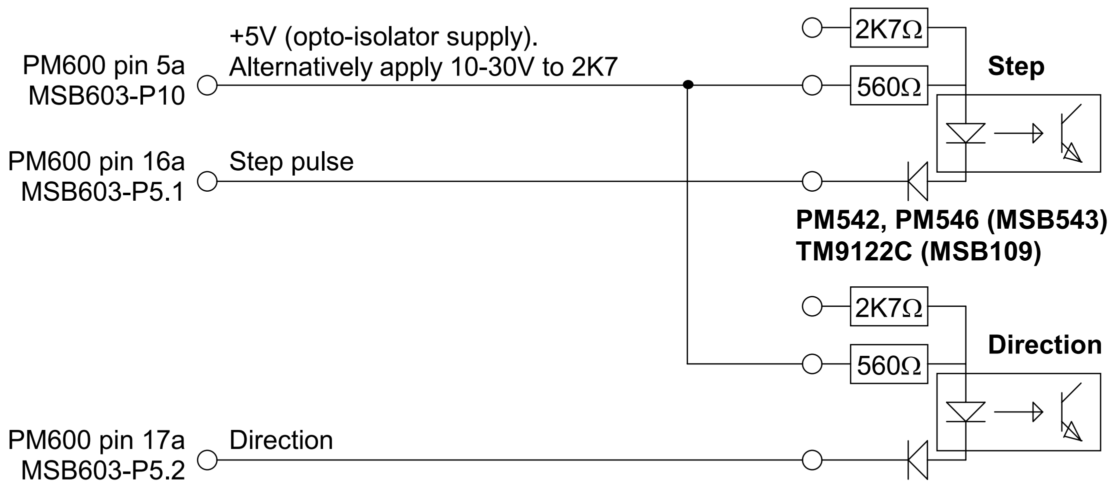

MSB603-P10.1 (pins 05a, 05b, 05c)
+5V Output
MSB603-P10.2 (pins 05a, 05b, 05c)
+5V Output

3.6 Clock Pulse And Direction Signal Outputs

The PM600 has open-collector outputs (rated at 30mA, 35V) for both the clock pulse (step) and the direction control signals. These signals are used as the command signals for a stepper drive.
The clock pulses will pull the output low for 10µS or 50% of the pulse rate; whichever is the shorter.
If the motor moves in the opposite direction to that required, then the direction can be corrected by swapping the polarity of one of the phases of the stepper motor (i.e. A+ and A- **or** B+ and B-). Alternatively on stepper drives such as the PM542 and PM546 a *reverse* switch on the drive can be used to reverse motor direction.

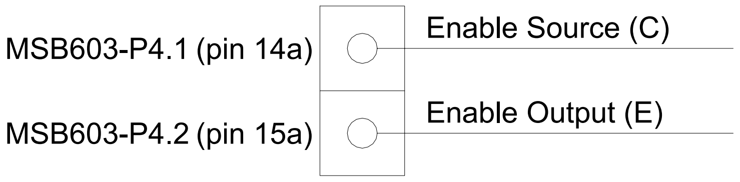


Example of connection to opto-isolated inputs such as those on the PM546, PM542 or TM9122 stepper drives.

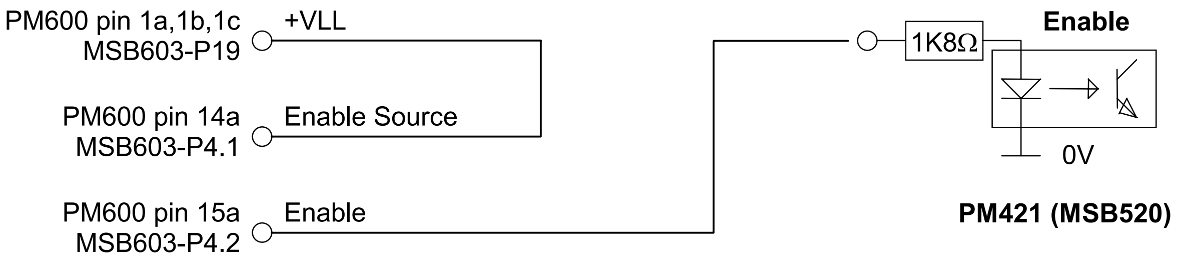


3.7 Enable Output

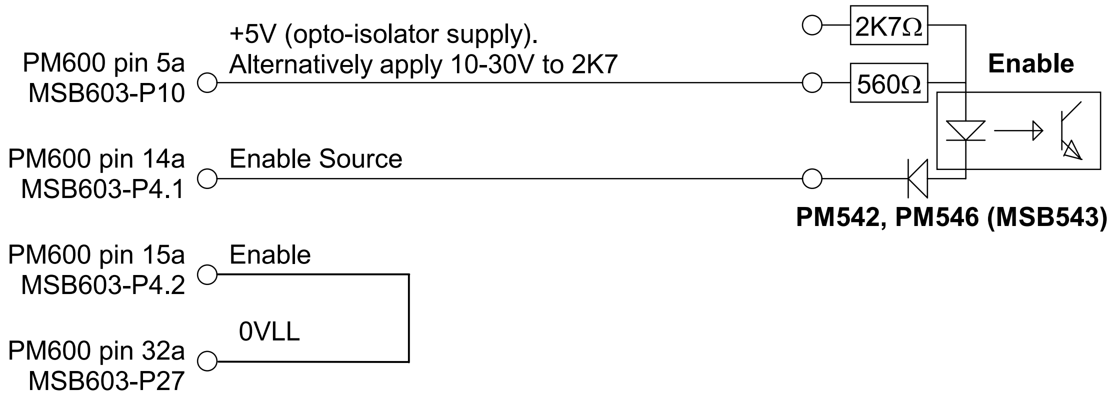
The Enable output is typically used in a system to enable either servo amplifiers such as the PM421 and PM810 or stepper drives such as the PM546 or MSE570. The Enable Source and Enable Output are connected to the Collector and Emitter of an opto-isolator NPN transistor. The output is rated at 50mA, 35V. The output must not be reverse biased and if used to drive an inductive load such as a relay then a diode must be used to protect the output.



Example of connection to inputs such as on the PM421 servo amplifier.



Example of connection to opto-isolated inputs such as on the PM546 and PM542 stepper drives.



The action of the enable output can be disabled by setting bit *h* of the *abort mode* using the **AM** command. If bit *h* is set to 1 the *enable output* is left ON during a control loop abort. If bit *h* is set to 0 (initial state) the *enable output* is switched OFF during a control loop abort to disable the drive.

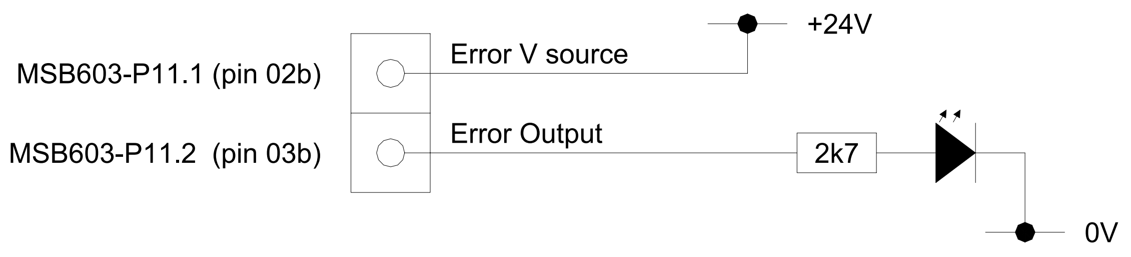
Caution: Do not disable fault detection features unless sure of operation.

3.8 Error Output

The Error output is activated at the same time as the front panel status display shows

Stop Input Abort	Command abort (**AB**)
Communication error	Controller failed self test
Stall error	Tracking error
Time Out error (not got there)	Communication error
Encoder quadrature error (Encoder 1 only)	Controller failed self test

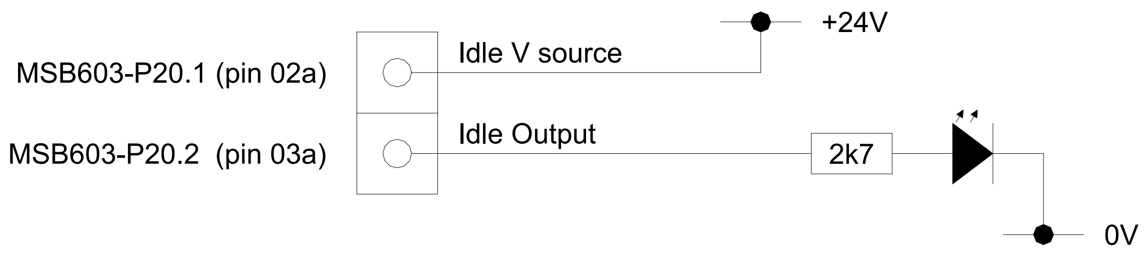
If the Error output is active then the Enable output will be turned off. The enable output is therefore the best output to connect to the motor drive. The output is rated at 50mA, 35V. The output must not be reverse biased and if used to drive an inductive load such as a relay then a diode must be used to protect the output.



3.9 Idle Output

The *idle* output is activated when the PM600 is *idle*. The front panel status display shows when the controller is *idle*.

The output is rated at 50mA, 35V. The output must not be reverse biased and if used to drive an inductive load such as a relay then a diode must be used to protect the output.



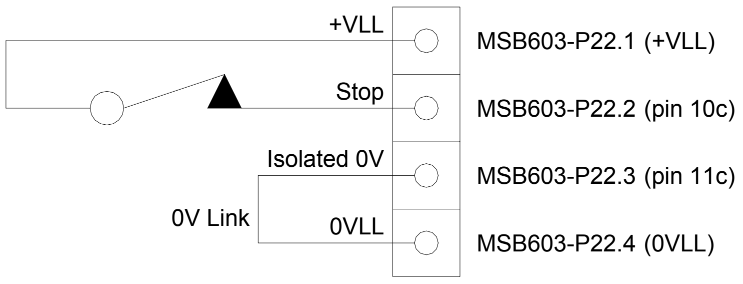
3.10 Input Isolation

The *Stop*, *Limits, Datum* and *Jog* inputs are opto-isolated. A +24V nominal supply must be used as the common for these inputs. This supply can either be a separate supply or the same supply as the PM600. If the same supply is used then the *Isolated 0V* terminals must be connected to the PM600 supply 0V. If a separate supply is used then the *Isolated 0V* terminals must be connected to the 0V terminal of the separate power supply and the +24V output from the separate supply can be used for the common of the switches.

The diagrams in this manual show the PM600 supply used as the isolated supply.

The input voltage that is considered as a logic 1 signal is 10-35V and the input voltage that is considered as a logic 0 is 0-5V.

3.11 Abort Stop Input



The Abort Stop input on the PM600 is activated by an opto-coupled input.

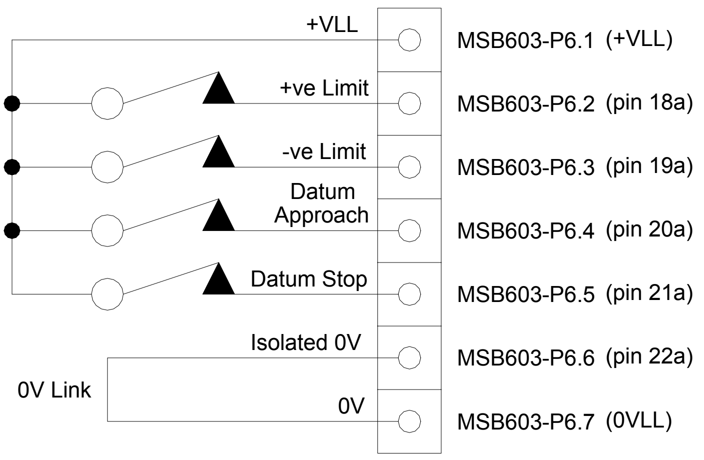
The switches used should be *normally-closed*. If the input is open-circuited an *input abort* is triggered, movement is stopped by setting the analogue output of the PM600 to 0V in servo mode or by interrupting the step clock and direction outputs in stepper modes. At the same time the enable output can be switched off (depending on the Abort Mode (**AM**) setting) and the error output is switched on.

All subsequent move commands are not acted upon. An *input abort* can be reset using the **RS** (reset) command. The unit can also be reset by powering-down. The response to a move command on axis 1 is **01:! INPUT ABORT**. Axis 1's response to a **CO** (current operation) command is **01:Input Abort**.

If the Stop Input is not used the input connection must be **made** for normal operation. The stop input should be connected to the +ve supply, and its isolated 0v to the supply 0v. If an MSB603 motherboard is being used, this can be achieved using 0.1" jumper links.

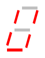
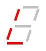
IMPORTANT - The Stop input on the PM600 must not be used in isolation as a safety stop and as such should only be treated as a monitoring device. For a correct safety stop the power must be removed from the motor.

3.12 Hard Limit and Datum Inputs



3.12.1 Hard Limits

The upper and lower hard limits are inputs to the PM600 which if activated prevent further movement of the motor in the appropriate direction. They are derived from normally-closed limit switches. In the open state, movement is disabled. If no limit switches are to be used, the limit input connections should be **made** to enable moves. On hitting a Hard Limit the motor will decelerate at the Limit Deceleration (**LD**) rate.
The Query All (**QA**) and Output Status (**OS**) commands can be used to check the action of the limit switches. Viewing the PM600's front panel status display will also check the action of the limit switches.

 Upper hard-limit activated  Lower hard-limit activated

3.12.2 Datum Inputs

With an incremental system, it is often necessary to find some datum point, so those moves are then relative to the physical position of the mechanism.

A Datum switch or sensor can be used. This is connected to the Datum Stop input. The Datum switch should be positioned so that it is activated as the load passes it and will therefore not be damaged.
A dot shows in the PM600's status display when a datum position is captured.

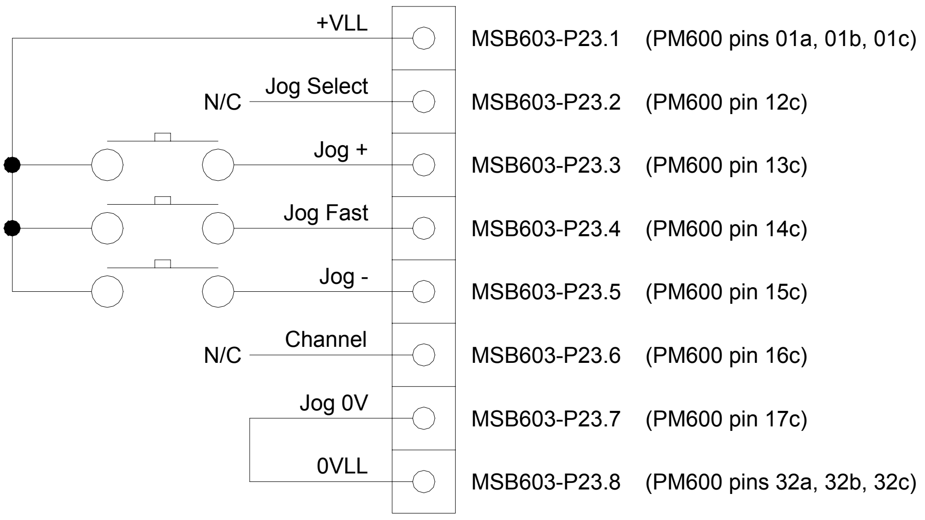
The Datum Approach input controls the speed during a home to datum (**HD**) move. Refer to Datum Search section.

3.13 Jog Controls

The PM600 controller is provided with manual *jog* control inputs, which can be used to allow the user to move the motor, perhaps to find an undefined position, which can then be identified by using the **OC** (output command position) command. There are three inputs that are activated by connecting to 10-35V. This can be achieved by using normally open switches e.g. on a jog box.
The Jog inputs are opto-coupled and operate when a switch is closed. Momentary closure of the Jog+ or Jog- switches produces a single step. If the Jog+ or Jog- switch is closed for more than 0.5 seconds the motor accelerates to the slow jog speed.
If the Fast Jog switch is then closed, the motor will accelerate to the fast jog speed. Opening the Fast Jog switch decelerates the motor to the slow rate until the Slow Jog switch is opened.

The jog speeds are set using the commands:

SJ set jog speed.


+VLL
MSB603-P23.1 (PM600 pins 01a, 01b, 01c)
N/C
Jog Select
MSB603-P23.2 (PM600 pin 12c)
Jog +
MSB603-P23.3 (PM600 pin 13c)
Jog Fast
MSB603-P23.4 (PM600 pin 14c)
Jog -
MSB603-P23.5 (PM600 pin 15c)
N/C
Channel
MSB603-P23.6 (PM600 pin 16c)
Jog 0V
MSB603-P23.7 (PM600 pin 17c)
0VLL
MSB603-P23.8 (PM600 pins 32a, 32b, 32c)

SF set fast jog speed.

The mode of operation for using jog switches must be set using the Jog Mode (**JM**) command.

Example:

1JM10010000	Set axis 1 to use Jog switches only (joystick and Input encoder jog are both disabled). Channel Increment is enabled.

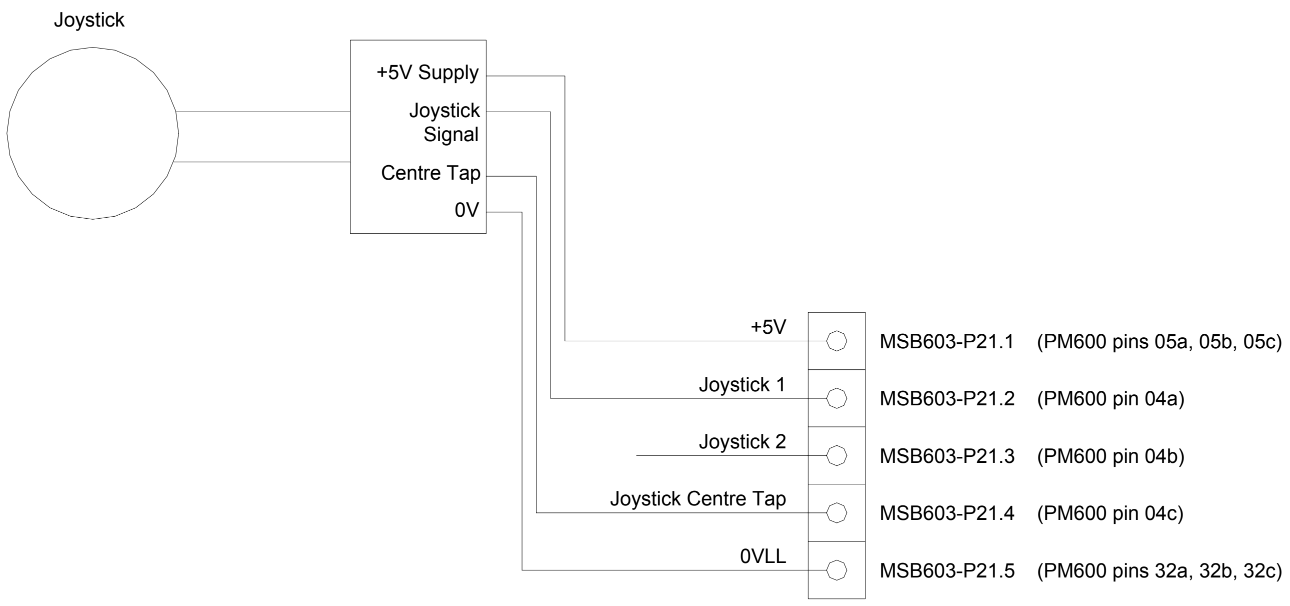
3.14 Joystick Controls

A Joystick can also be used to control the motors. There are various commands that are used to calibrate the operation of the joystick control.

JC	Joystick centre – used to set the zero point of the joystick.
JR	Joystick range – used to set the range of the joystick.
JS	Joystick speed – used to set the maximum speed when using joystick control.
JT	Joystick Threshold – used to set the amount of joystick movement before any move occurs.
QJ	Query the current Joystick settings.

The mode of operation for using a joystick must also be set using the **JM** command.
Example:

1JM01000000	Set axis 1 to use a joystick only (Jog switches and Input encoder jog are both disabled).


Joystick
+5V Supply
Joystick Signal
Centre Tap
0V
+5V
Joystick 1
Joystick 2
Joystick Centre Tap
0VLL
MSB603-P21.1 (PM600 pins 05a, 05b, 05c)
MSB603-P21.2 (PM600 pin 04a)
MSB603-P21.3 (PM600 pin 04b)
MSB603-P21.4 (PM600 pin 04c)
MSB603-P21.5 (PM600 pins 32a, 32b, 32c)

3.15 Encoder Inputs

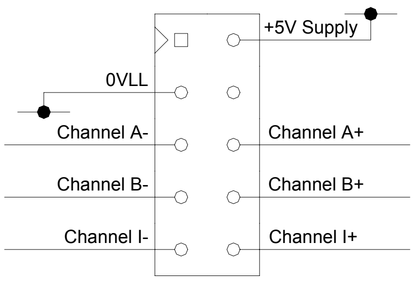
The PM600 has inputs for either TTL output (or *sinking* open collector output) or RS422 complementary output type. The regulated +5 volt output can be used to power the encoders. The leads to the encoders should be screened, with the screen grounded. If RS422 connections are used, *twisted-pair* cable should be used. If TTL encoders are used then the connections are made to the A+, B+ and I+ inputs.

There are three encoder inputs.

1. The Position Encoder (Encoder 1) is used to measure the Position of either the load or the motor.
2. The Secondary (Auxiliary) encoder (Encoder 2) is fitted to a servo motor. It is used as a damping term in the servo loop.
3. The Master encoder (Encoder 3) is used to synchronise the PM600 to another mechanism.

3.15.1 Encoder 1 – Position Encoder

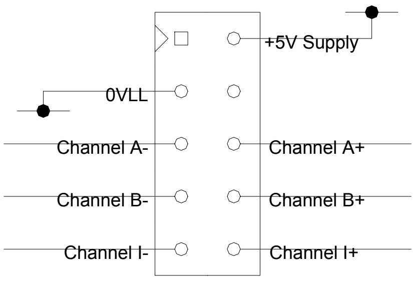
The position encoder can either be mounted on the motor's rear shaft or remotely from the motor on the mechanism.
Mounting the position encoder on the mechanism and thereby measuring the output of the system is the most accurate method, eliminating any lost motion in the mechanism. However the stability of the system is more difficult to achieve due to this lost motion. In a servo system this can be compensated for by using an auxiliary encoder (encoder 2) on the rear shaft of the motor.
Ensure that the sense of direction of the motor's encoder corresponds to that of the motor. If it is found that the encoder position changes in the reverse sense to the movement, then this can be corrected by either swapping encoder connections A+ and A- with B+ and B- or by setting a negative encoder ratio.
E.g. 1ER-1/1.
The **OA** command can be used to find the value of the *actual position* generated by Encoder 1.



MSB603-P2.1		(15 way D pin 1)	MSB603-P2.2	(+5V)	(15 way D pin 9)
MSB603-P2.3	(0VLL)	(15 way D pin 2)	MSB603-P2.4		(15 way D pin 10)
MSB603-P2.5	(PM600 pin 07a)	(15 way D pin 3)	MSB603-P2.6	(PM600 pin 06a)	(15 way D pin 11)
MSB603-P2.7	(PM600 pin 09a)	(15 way D pin 4)	MSB603-P2.8	(PM600 pin 08a)	(15 way D pin 12)
MSB603-P2.9	(PM600 pin 11a)	(15 way D pin 5)	MSB603-P2.10	(PM600 pin 10a)	(15 way D pin 13)

3.15.2 Encoder 2 - Auxiliary Encoder

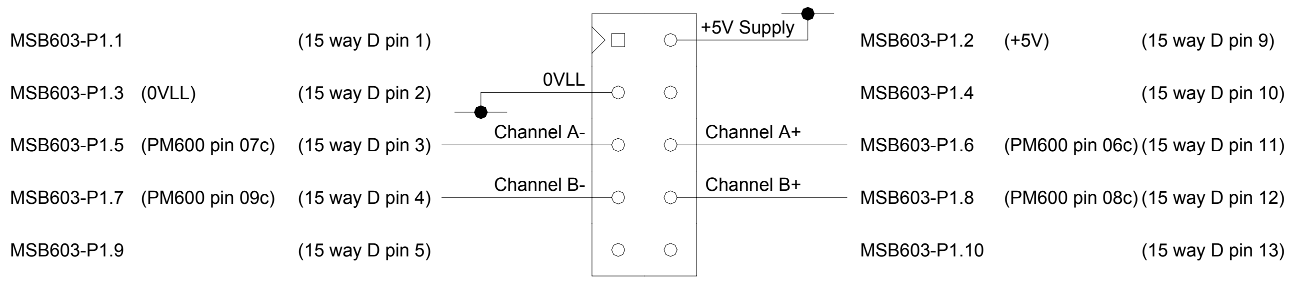
Ensure that the sense of direction of the Auxiliary (motor) encoder corresponds to that of the position encoder. If it is found that the encoder position changes in the reverse sense to the movement, then this can be corrected by swapping encoder connections A+ and A- with B+ and B-.
The **OT** command can be used to find the value of the position generated by Encoder 2.



MSB603-P13.1		(15 way D pin 1)	MSB603-P13.2	(+5V)	(15 way D pin 9)
MSB603-P13.3	(0VLL)	(15 way D pin 2)	MSB603-P13.4		(15 way D pin 10)
MSB603-P13.5	(PM600 pin 07b)	(15 way D pin 3)	MSB603-P13.6	(PM600 pin 06b)	(15 way D pin 11)
MSB603-P13.7	(PM600 pin 09b)	(15 way D pin 4)	MSB603-P13.8	(PM600 pin 08b)	(15 way D pin 12)
MSB603-P13.9	(PM600 pin 11b)	(15 way D pin 5)	MSB603-P13.10	(PM600 pin 10b)	(15 way D pin 13)

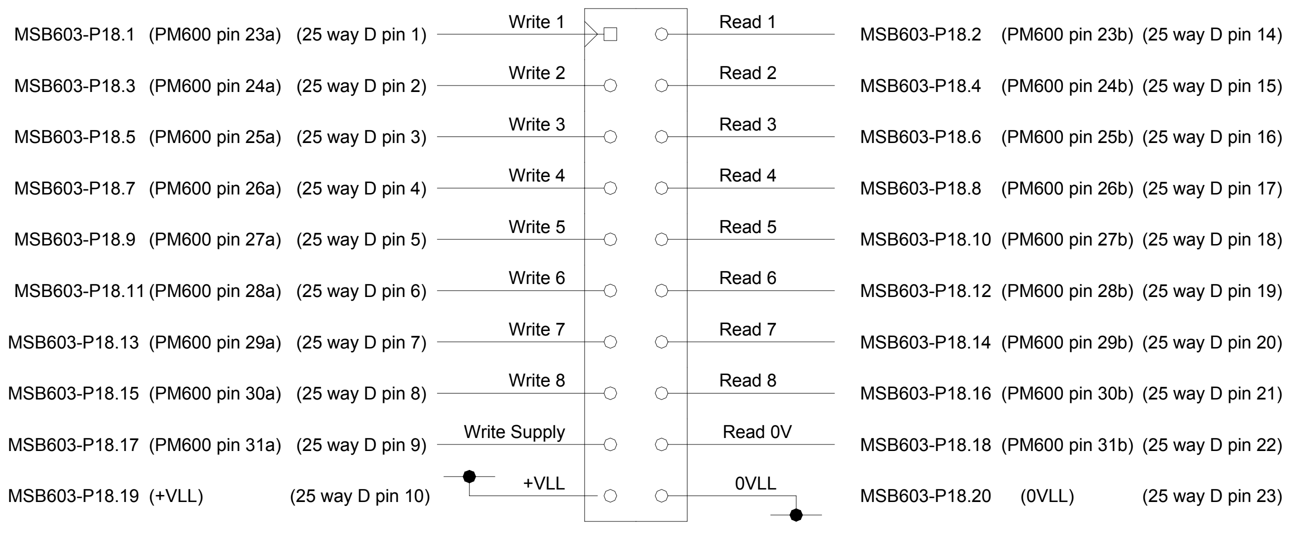
3.15.3 Encoder 3 – Master Gearbox Encoder or Jogging Encoder

The sense of direction of the master encoder can be reversed by either swapping encoder connections A+ and A- with B+ and B- or by setting a negative gearbox ratio. E.g. 1GR-1/1.
The **OI** command can be used to find the value of the *input position* generated by Encoder 3.



MSB603-P1.1		(15 way D pin 1)		+5V Supply	MSB603-P1.2	(+5V)	(15 way D pin 9)
MSB603-P1.3	(0VLL)	(15 way D pin 2)	0VLL		MSB603-P1.4		(15 way D pin 10)
MSB603-P1.5	(PM600 pin 07c)	(15 way D pin 3)	Channel A-	Channel A+	MSB603-P1.6	(PM600 pin 06c)	(15 way D pin 11)
MSB603-P1.7	(PM600 pin 09c)	(15 way D pin 4)	Channel B-	Channel B+	MSB603-P1.8	(PM600 pin 08c)	(15 way D pin 12)
MSB603-P1.9		(15 way D pin 5)			MSB603-P1.10		(15 way D pin 13)

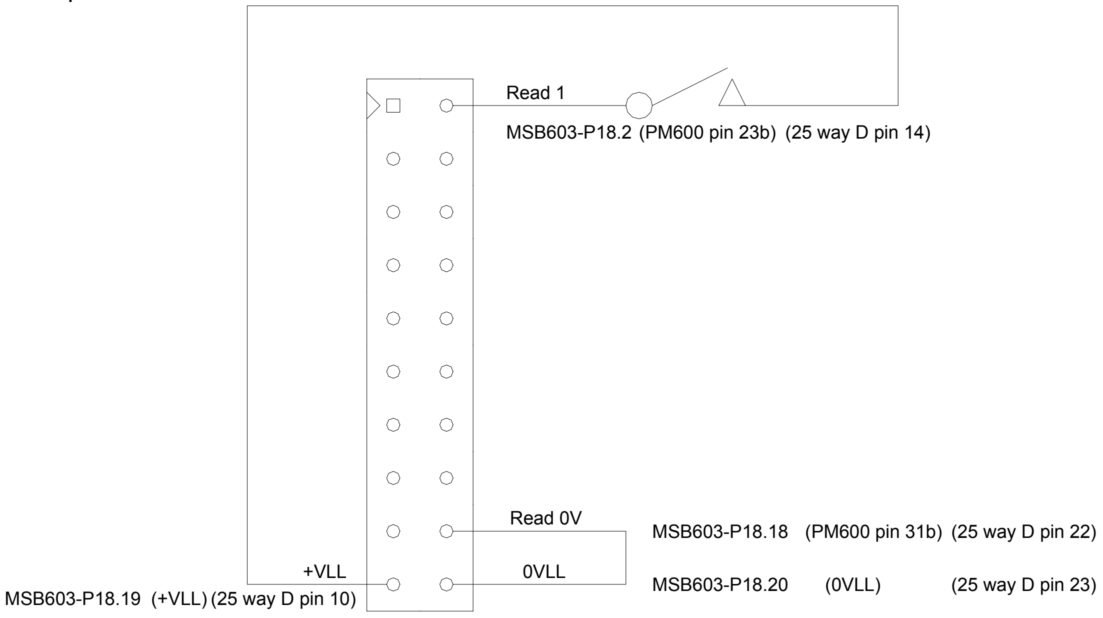
3.16 Digital Inputs And Outputs



MSB603-P18.1	(PM600 pin 23a)	(25 way D pin 1)	Write 1	Read 1	MSB603-P18.2	(PM600 pin 23b)	(25 way D pin 14)
MSB603-P18.3	(PM600 pin 24a)	(25 way D pin 2)	Write 2	Read 2	MSB603-P18.4	(PM600 pin 24b)	(25 way D pin 15)
MSB603-P18.5	(PM600 pin 25a)	(25 way D pin 3)	Write 3	Read 3	MSB603-P18.6	(PM600 pin 25b)	(25 way D pin 16)
MSB603-P18.7	(PM600 pin 26a)	(25 way D pin 4)	Write 4	Read 4	MSB603-P18.8	(PM600 pin 26b)	(25 way D pin 17)
MSB603-P18.9	(PM600 pin 27a)	(25 way D pin 5)	Write 5	Read 5	MSB603-P18.10	(PM600 pin 27b)	(25 way D pin 18)
MSB603-P18.11	(PM600 pin 28a)	(25 way D pin 6)	Write 6	Read 6	MSB603-P18.12	(PM600 pin 28b)	(25 way D pin 19)
MSB603-P18.13	(PM600 pin 29a)	(25 way D pin 7)	Write 7	Read 7	MSB603-P18.14	(PM600 pin 29b)	(25 way D pin 20)
MSB603-P18.15	(PM600 pin 30a)	(25 way D pin 8)	Write 8	Read 8	MSB603-P18.16	(PM600 pin 30b)	(25 way D pin 21)
MSB603-P18.17	(PM600 pin 31a)	(25 way D pin 9)	Write Supply	Read 0V	MSB603-P18.18	(PM600 pin 31b)	(25 way D pin 22)
MSB603-P18.19	(+VLL)	(25 way D pin 10)	+VLL	0VLL	MSB603-P18.20	(0VLL)	(25 way D pin 23)

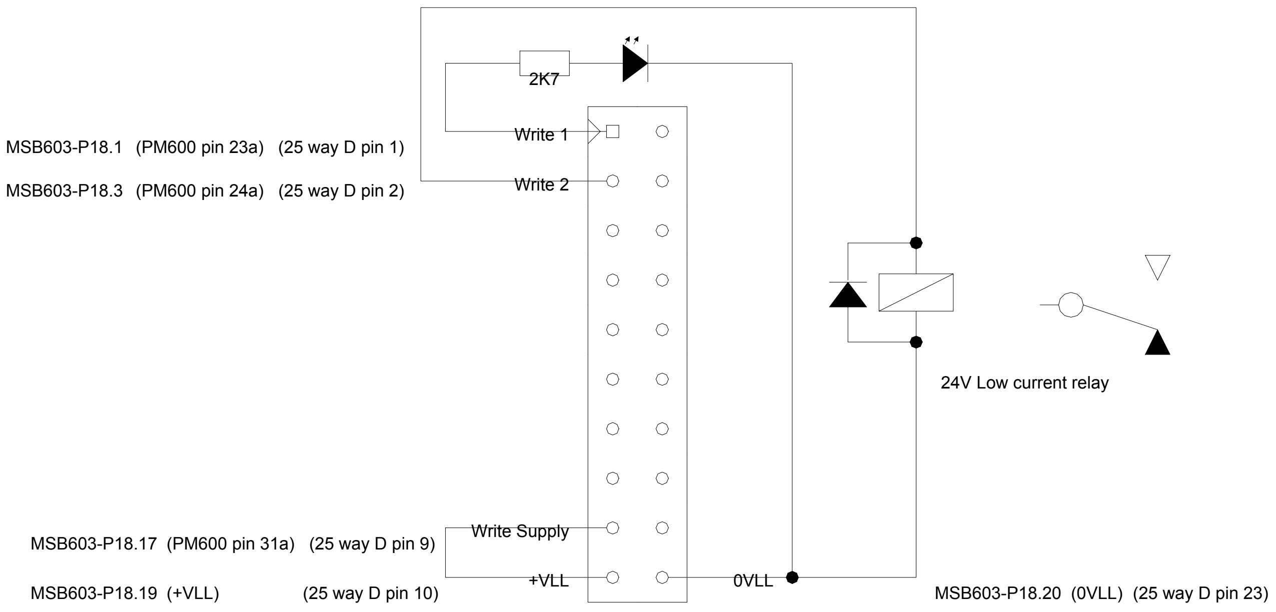
3.16.1 Read Ports

The PM600 controller has eight input ports, known as *read ports* 1 to 8. These inputs are opto-coupled and are activated by an input voltage of between +10V and +35V (**1**). If a *read port* is either not connected or below +5V it will read as **0**.
The input can be connected to either a PNP signal output, a switch, or another controller's output *write port*.
The **RP** Read input Port command is used to check the operation of the *read ports*. This instruction returns an eight digit binary number of either **0** or **1** characters to represent the current state of the read port. These start with read port 8, through to 1. Referring to the diagram below, a **1** represents a closed switch and a **0** represents an open switch.

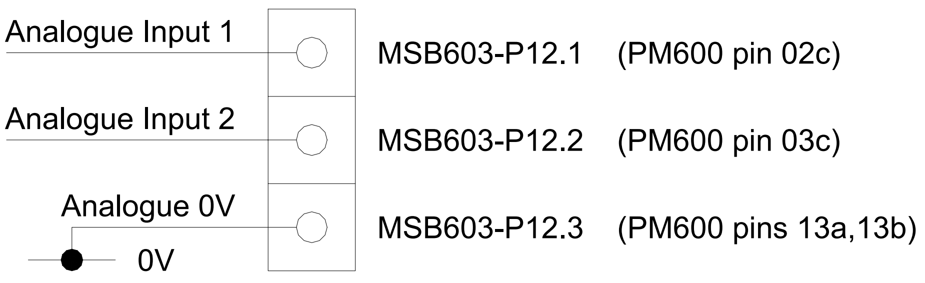

Read 1
MSB603-P18.2 (PM600 pin 23b) (25 way D pin 14)
Read 0V
MSB603-P18.18 (PM600 pin 31b) (25 way D pin 22)
+VLL
0VLL
MSB603-P18.20 (0VLL) (25 way D pin 23)
MSB603-P18.19 (+VLL) (25 way D pin 10)

3.16.2 Write Ports

The PM600 controller has eight output ports, known as *write ports* 1 to 8. These outputs are driven by opto-couplers. The outputs can be connected to an indicator (LED) opto-isolator or another controller's input *read port*. The **WP(bit pattern)** command is used to write to the output port.
The bit pattern is specified as a eight digit binary number. The digits will be either characters **0, 1** or **2** starting with *write port* 8 through to 1. A **0** defines that the output will be **off**, a **1** defines that the output will be **on** and a **2** defines that the output will not change from its current state. The power-on states of the *write ports* are **00000000** - i.e. all **off**. The outputs are rated at 50mA, 35V. They must not be reverse biased and if used to drive an inductive load such as a relay then a diode must be used to protect the output.

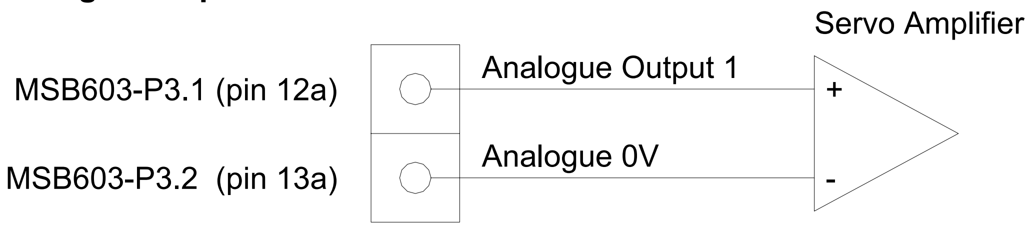

2K7
MSB603-P18.1 (PM600 pin 23a) (25 way D pin 1)
Write 1
MSB603-P18.3 (PM600 pin 24a) (25 way D pin 2)
Write 2
24V Low current relay
MSB603-P18.17 (PM600 pin 31a) (25 way D pin 9)
Write Supply
MSB603-P18.19 (+VLL) (25 way D pin 10)
+VLL
0VLL
MSB603-P18.20 (0VLL) (25 way D pin 23)

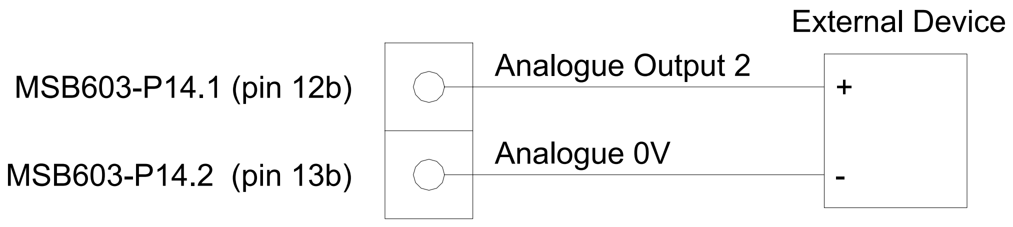
3.16.3 Analogue Inputs


Analogue Input 1
Analogue Input 2
Analogue 0V
0V
MSB603-P12.1 (PM600 pin 02c)
MSB603-P12.2 (PM600 pin 03c)
MSB603-P12.3 (PM600 pins 13a,13b)

There are five analogue inputs. Inputs 1 and 2 are general purpose bipolar inputs and 3 to 5 are for use with Joysticks (see Joystick Controls) although they can be used as general purpose unipolar inputs.

3.16.4 Analogue Outputs



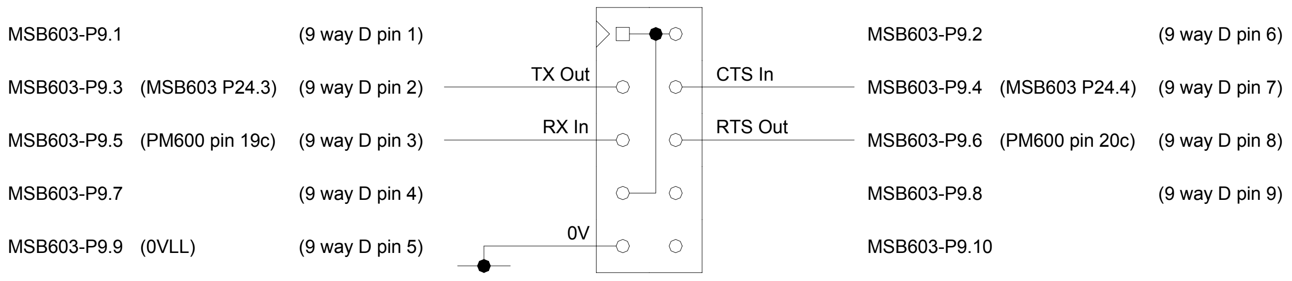


There are two analogue outputs. Analogue output 1 is used by the PM600 in servo mode to drive a servo amplifier although it can be used as general purpose output in stepper control modes. Both analogue outputs are bipolar.

3.17 Serial Interface (Comms) Connections

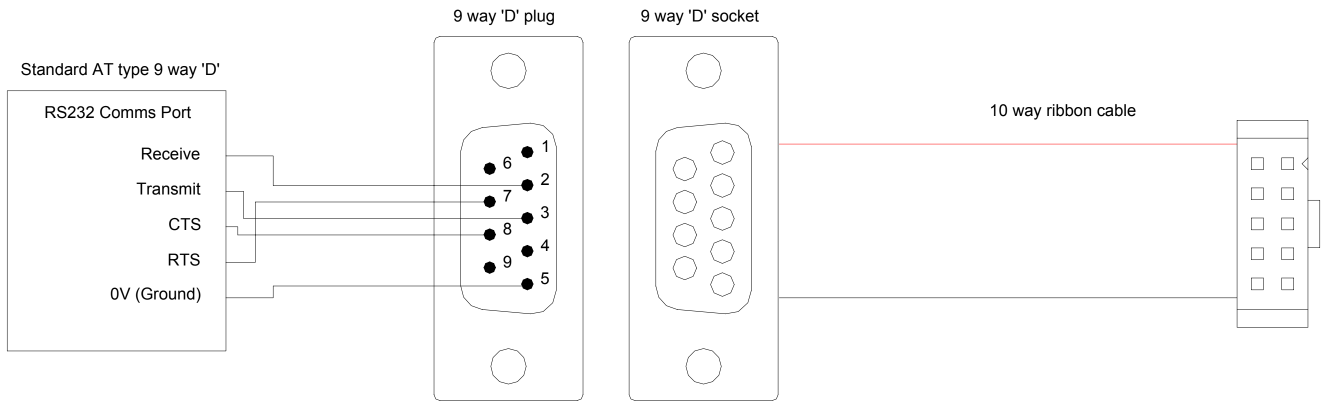
The PM600 commands are received by either the RS232 port or the RS485 port as strings of characters. The RS232 is configured in a three format or a five wire format using CTS and RTS handshaking.

3.17.1 Primary Comms Connections


MSB603-P9.1 (9 way D pin 1)
MSB603-P9.3 (MSB603 P24.3) (9 way D pin 2)
MSB603-P9.5 (PM600 pin 19c) (9 way D pin 3)
MSB603-P9.7 (9 way D pin 4)
MSB603-P9.9 (0VLL) (9 way D pin 5)
TX Out
RX In
0V
CTS In
RTS Out
MSB603-P9.2 (9 way D pin 6)
MSB603-P9.4 (MSB603 P24.4) (9 way D pin 7)
MSB603-P9.6 (PM600 pin 20c) (9 way D pin 8)
MSB603-P9.8 (9 way D pin 9)
MSB603-P9.10

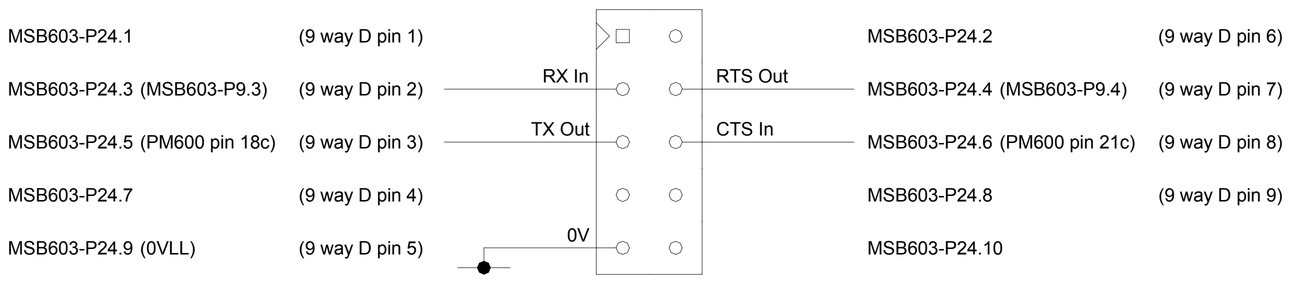
Serial comms connections to the MSB603 can be connected via a 9 way 'D' connector as shown in the following example.

Example: RS232 Control from a PC

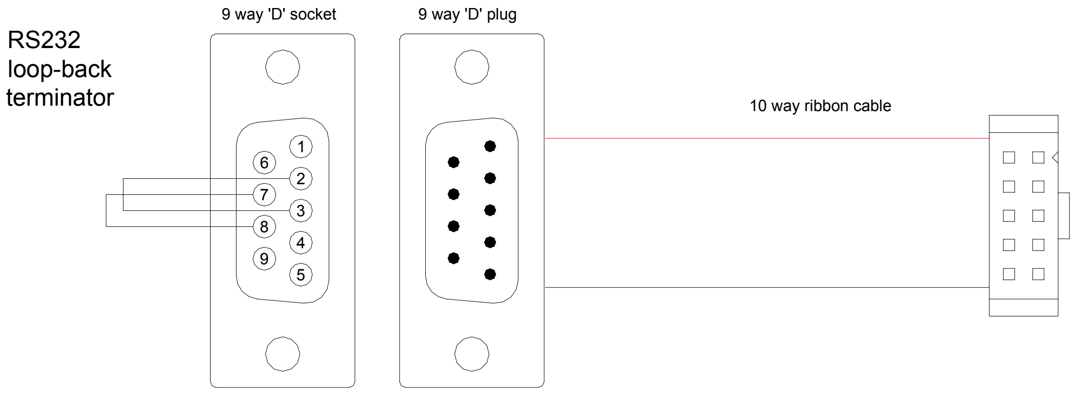


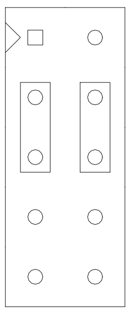
The ribbon cable socket can be plugged directly into a serial port on a PC.

3.17.2 Secondary Comms Connections

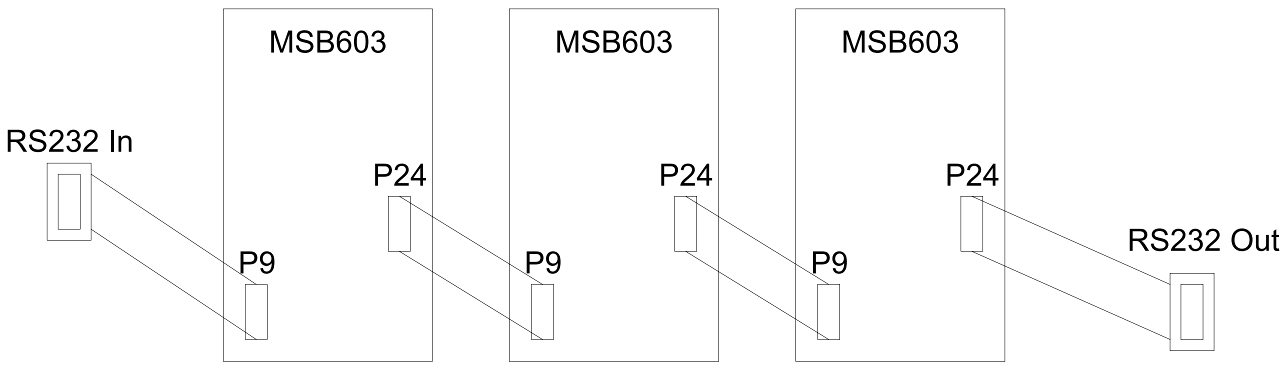

MSB603-P24.1 (9 way D pin 1)
MSB603-P24.3 (MSB603-P9.3) (9 way D pin 2)
MSB603-P24.5 (PM600 pin 18c) (9 way D pin 3)
MSB603-P24.7 (9 way D pin 4)
MSB603-P24.9 (0VLL) (9 way D pin 5)
RX In
TX Out
0V
RTS Out
CTS In
MSB603-P24.2 (9 way D pin 6)
MSB603-P24.4 (MSB603-P9.4) (9 way D pin 7)
MSB603-P24.6 (PM600 pin 21c) (9 way D pin 8)
MSB603-P24.8 (9 way D pin 9)
MSB603-P24.10

The Secondary Serial comms connections to the MSB603 can be connected via a 9 way ‘D’ connector as shown in the following example. The Secondary comms connector is used for connecting either to a subsequent control rack or to an RS232 loop-back terminator. The RS232 Loop-Back Terminator must be fitted to allow both replies from the controllers and commands sent to the controllers to be echoed back.

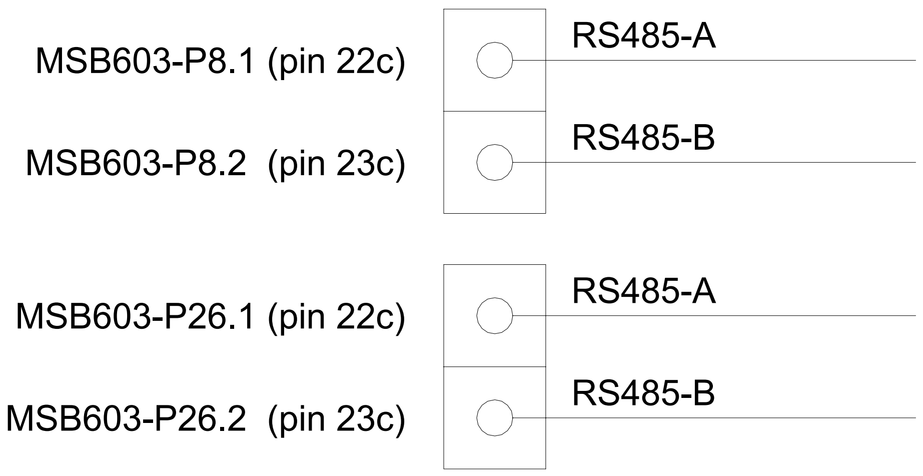



Alternatively 0.1” links can be fitted directly to MSB603 - P24.

3.17.3 Multi-Axis Communication



On multi-axis systems, use a ribbon cable to connect between adjacent MSB603 motherboards. MSB603-P24 on one axis connects to MSB603-P9 on the next axis.



3.17.4 RS485 Connections

In systems using RS485 communications, the RS485 bus can be parallel connected to the PM600. Alternatively if an MSB603 motherboard is being used the connections can be made to P8 and P26.

A 100Ω terminator should be fitted to the end of the RS485 bus. This can be done by turning on the terminator on the last PM600 on the bus using SW3-8.

RS485 terminator select

8 ■□	8 ■□ No terminator	8 □■ 100Ω terminator

The PM600 has no biasing resistors fitted on board, so they must be fitted externally.

Note that in RS485 mode, received characters are not echoed back.

4 Commissioning

4.1 Minimum Wiring Requirements

The minimum wiring requirements to run a system are:

Power Supply to controller.
Abort Stop switch or defeating link.
Hard Limit switches or defeating links.
RS232 communication wiring.

4.1.1 Servo Control Mode

-10V to +10V signal to servo drive.
Servo Drive Power supply.
Servo Drive to motor wiring.

4.1.2 Stepper Control Modes

Clock and Direction signals to stepper drive.
Stepper Drive Power supply.
Stepper Drive to motor wiring.

On switching the unit on, the front panel STATUS display should show - (Idle) after about 3 seconds. If it shows A , then check the Stop Input, if U or L then check the appropriate hard-limit switch.
If a F is displayed then the PM600 Controller has failed its self test and must be returned for repair.

4.2 Checking Operation

4.2.1 Communications

The axis address of each PM600 must be set up with rotary switches SW1 and SW2.

The PM600 should be connected to an RS232 terminal or a PC running a terminal emulation such as Procomm™. The default RS232 setting of the PM600 is 9600 baud, 7 data bits, even parity and one stop bit (9600,7,E,1). Altering the settings of the Communication Configuration Switch SW3 can change this (see section 9.1). If an Γ is shown on the front panel status display then a communication error has occurred. The settings of both controllers and computer must be checked. In multiple axis applications, each controller should be set to a different address but all axes must be set to the same communication configuration.

Assuming axis one is being communicated with, send a **1ID** (identify) command. A response of typically: **01:Mclennan Digiloop Motor Controller V5.19a(0.6): Servo mode** should be received. If required send the command **1CM11** to change the control mode to open-loop stepper mode or **1CM1** to change to servo control mode. Note that a *Command Abort* will occur when changing between servo and stepper control modes. This may reset by sending a **1RS** command.

4.2.2 Abort Stop Input

If the *Abort Stop* input is not used, it must be linked. Activating the Stop input momentarily will latch the *Abort Stop* condition and the front panel STATUS display will show A indicating Stop Input Abort. If you attempt a move on axis 1 by a **1MR10** command, the controller will respond with a **01:! INPUT ABORT** error message. Sending a **CO** (current operation) command will also check if the stop input has been activated. If the Stop input has been activated on axis 1, a response of **01:Input Abort** will be received. This condition is reset with the **RS** command.
The operation of the **Stop** input can be selected to be momentary by the Abort Mode (**AM**) command.

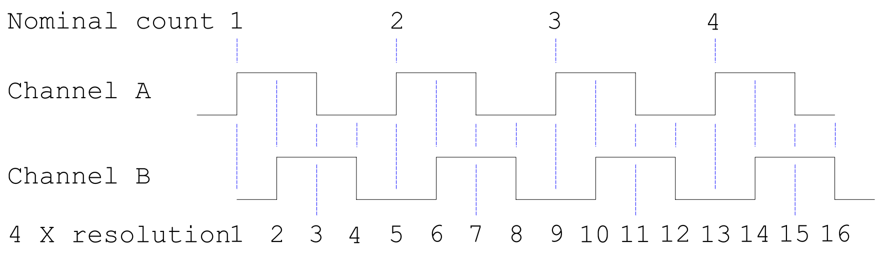
4.2.3 Hard Limits

If the **Hard Limit** inputs are not used, they must be linked. The status of the limits can be queried with the Output Status (**OS**) command, which will indicate that a hard limit is activated. Whilst either hard limit is activated, the front panel STATUS display will show if the Upper hard-limit (or both limits) is activated and if the Lower hard-limit is activated.

4.3 Servo Mode Commissioning

4.3.1 Checking Encoder Feedback

This test should be done with the motor disconnected or disabled.
The resolution of the encoder is multiplied by four since all transitions of the quadrature signals are counted:



The correct operation of the position feedback encoder must be established before any parameters are set. If possible move the encoder a known number of steps (say one revolution) then send a **1OA** (axis one - output actual position) command. A response of the number of steps moved should be received. The test should be repeated in the opposite direction. Remove power from the PM600.

The next stage should be attempted with caution.

The motor can now be reconnected or enabled. A relative move of 2000 steps should be sent using a **1MR2000** command. If the motor drives continuously until a shows on the PM600's STATUS display, then the motor movement direction compared with the encoder rotation sense is incorrect. If the motor moves towards its position then the encoder rotation sense is correct. If the motor does not move, increase the value of **KP** in the order 20, 50, 100, 200, etc. A **1OF** (axis one - output position following error) command can be sent to find the position error remaining.

WARNING:
A serious situation can arise if the feedback from the position encoder fails. A common reason for this is the accidental disconnection of a plug carrying the encoder signals. When this happens the PM600 believes the motor to be stationary and applies full drive voltage to the motor in order to correct a supposed position error. A stalled motor (or encoder failure) is detected by looking for changes in the position encoder signals (or equivalently the changes in observed motor position). If the position encoder does not move, and the voltage output from the PM600 exceeds the value set by the **TH** (threshold) command for a time of 256ms, then the PM600 will set its output to zero and set a *Motor Stalled* condition. The front panel status display will also show .

4.3.2 Movement Direction

If the direction of operation needs to be reversed, either swap encoder connections A+ with B+ and A- with B-, or use a negative encoder ratio (**ER**) and swap both the motor polarity and the tacho (if used) polarity.

4.3.3 Setting The Servo Loop Coefficients

The values for **KP**, **KS**, **KV KF** and **KX** determine the characteristic of the servo loop, and will need to be established (even if only roughly) before proceeding further. See section 5. The coefficient **KX** is **only** used in the *Dual Encoder Mode*.

An approximate set of coefficients can usually be derived by invoking the **TUNE** command.

The tuning algorithm may fail if there is excessive backlash, or if the low frequency loop gain is very small or very large. Further optimisation of system response will be required to achieve the desired performance.

4.3.4 Setting Tracking Abort

The value of **TR** (tracking window) should be set as the maximum permissible error between *command position* and *actual position*. If an error between the *command position* and the *actual position* exceeds this value, a tracking abort occurs. The error between *actual* and *command* positions can be found using the **OF** (output following error) command. The value of **TR** should be set high enough to avoid nuisance triggering, but low enough to detect a system failure. If a long move or a constant velocity move (**CV**) is executed, the *following error* can be found by repeatedly sending an **OF** command and examining the replies. The magnitude of the *following error* will increase during the acceleration and deceleration phases of a move, so this should be taken into account when setting **TR**. The default value of **TR** is 4000 steps; this equates to one revolution of the motor when using a motor fitted with a 1000 line encoder.

The controller can be set to ignore a *tracking abort* condition by setting the abort mode bit *d* to ‘1’ using the abort mode (**AM**) command. The tracking abort function can be reinstated using the abort mode command, setting the abort mode bit *d* to ‘0’.

Caution: Do not disable fault detection features unless sure of operation.

A *tracking abort* can be reset by sending an **RS** (reset abort) command.

4.3.5 Setting Stall Threshold

The value of **TH** will set the motor stalled threshold. Failure of an encoder is indistinguishable from a stalled motor, and messages from the PM600 refer to *motor stalled* rather than encoder failure.

A stalled motor (or encoder failure) is detected by looking for changes in the position encoder signals. If the motor does not move, the voltage output value from the PM600 will increase until exceeds the value set by the **TH** command for a time of 256ms. The PM600 will then set its output to zero and set a Motor Stalled condition.

The servo system will have coulomb friction and the voltage required to overcome this friction, varies from system to system. The value of **TH** must therefore be large enough not to *nuisance trigger* but small enough to detect any failure.

The stall detection can be turned off for commissioning or difficult systems by setting the abort mode bit *c* to ‘1’ using the **AM** (abort mode) command. The stall abort function can be reinstated using the **AM** (abort mode) command, setting the abort mode bit *c* to ‘0’.

Caution: Do not disable fault detection features unless sure of operation.

A stall abort can be reset by sending an **RS** (reset abort) command.

4.4 Stepper Mode Commissioning

4.4.1 Movement Direction

Start with open-loop stepper mode by selecting with the command **CM11**. If changing from servo mode the PM600 will execute a *command abort* that can be reset by sending an **RS** command. A small test move can be attempted by a **MR10** command, to ensure the motor moves in the required direction. If it moves in the wrong direction, then either turning on the reverse switch on the drive (PM542 & PM546) or swapping the connections of one phase of the motor will reverse the direction.

4.4.2 Setting Up The Encoder (Closed-loop Control Modes Only)

As in the servo control mode each edge of the quadrature signals is counted, so the number of encoder counts per revolution will be four times the line count of the encoder.

The incoming encoder pulses are scaled by the encoder ratio (**ER**) formed by the Encoder Numerator (**EN**) and the Encoder Denominator (**ED**) i.e. the number of pulses received is multiplied by the numerator and divided by the denominator, to become the Actual Position.

If the ratio between the number of steps and the encoder counts is 1:1 (e.g. 100 line encoder = 400 counts per revolution mounted on a 400 step per rev motor) then the encoder scaling factors; Encoder Numerator and Encoder Denominator can be left at their default of 1, otherwise they will need to be changed.

If the Encoder Numerator (**EN**) is set to the number of motor steps per revolution and the Encoder Denominator (**ED**) is set to the number of encoder counts per revolution, then the ratio will be correct. E.G. If a 2000 step per revolution motor/microstep drive combination fitted with a 1024 line (4096 counts) encoder, then set the Numerator to 2000 and the Denominator to 4096 (**ER2000/4096** or **EN2000, ED4096**).

On switch on, both the Command Position and the Actual Position will both be at zero. Try a test move (e.g. **MR1000**) and query the positions with the **OC** and **OA** (or **QP**) commands, to ensure the scaling is correct. If one is the negative of the other, then the 'sense of direction' needs to be reversed, by either swapping the A and B signals OR by setting the Encoder Numerator to a negative value.

4.4.3 Closed-loop Stepper Mode Selection

Once the encoder is setup the appropriate closed-loop control mode can be selected using the **CM** command.

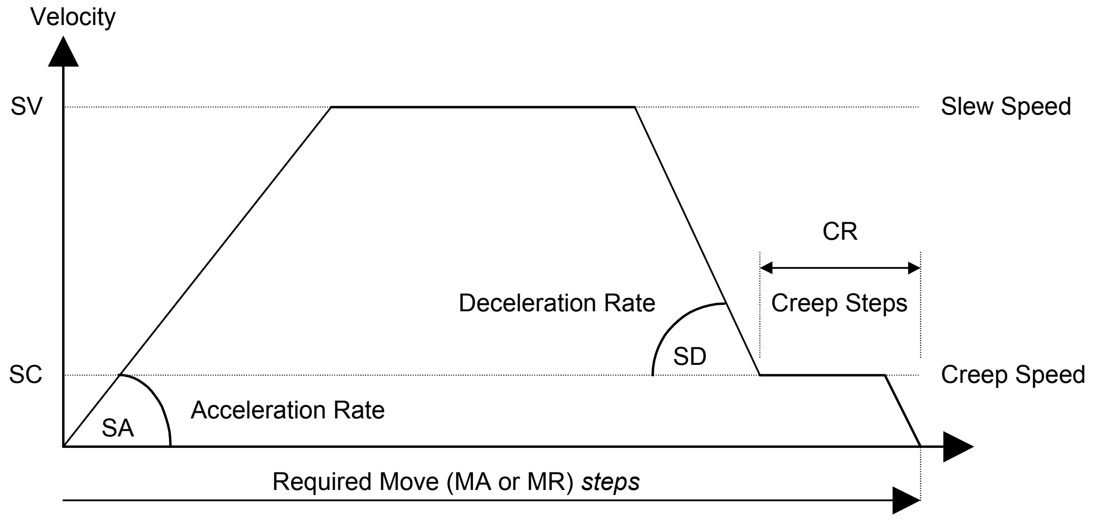
CM12 selects Checking Stepper Mode. Its operation is the same as open-loop, but the encoder fitted to the motor or mechanism monitors its actual position. A *tracking error* such as a motor stall can be detected. At the end of a move the *end of move* criterion is checked but any error is not corrected.

CM14 selects Closed-loop Stepper mode. Its operation is the same as open-loop, but the encoder fitted to the motor or mechanism monitors its actual position. A *tracking error* such as a motor stall can be detected. At the end of a move the *end of move* criterion is checked and any position error is automatically corrected.

CM13 selects External loop stepper mode. In a system consisting of a brushless motor fitted with rotation sensor, a drive and a PM600 controller, the operation is similar to open-loop but with the drive correcting for any position errors. The actual position of the motor can be monitored by the encoder or resolver fitted to the motor.

4.5 Parameter Setting

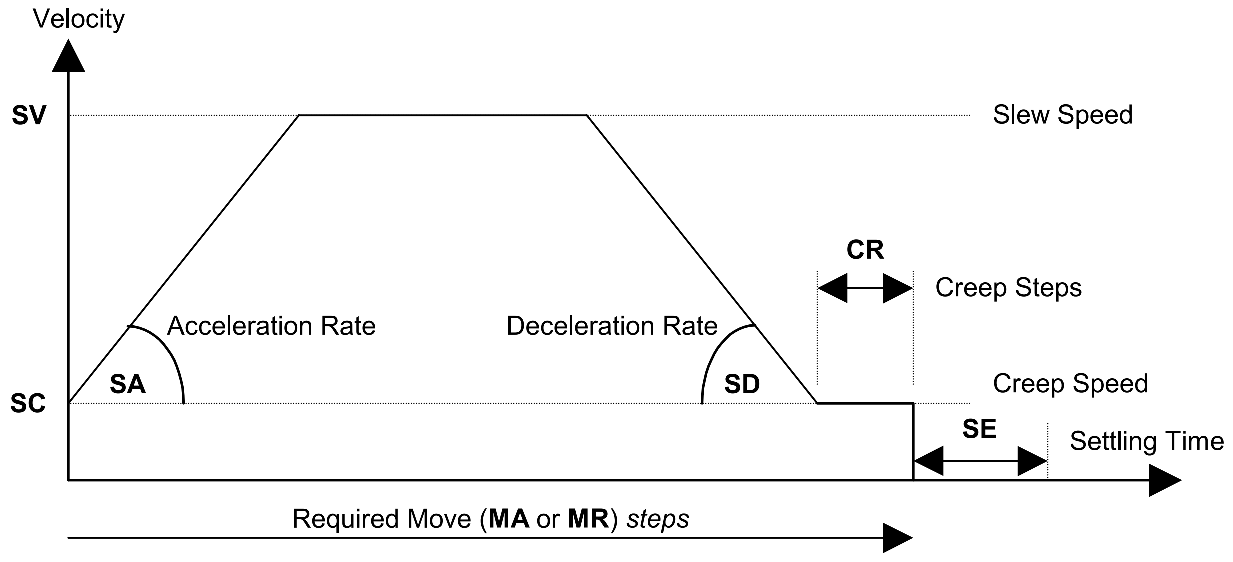
4.5.1 Setting Speeds And Acceleration/Deceleration In Servo Mode



The slew speed is set using the **SV** command. The initial value for **SV** is 1000 steps per second, which with a servo motor fitted with a 1000 line encoder equates to 15 r.p.m. This speed is usually too slow for the intended use. If the value of **SV** is set greater than maximum speed attainable by the motor, then an error between the command position and the actual position will build up and a tracking error will occur.

The acceleration setting **SA** and the deceleration setting **SD** can initially be set to give say a 0.5 Sec. ramp time. For example if the value of **SV** is 10000 steps/sec then the values of **SA** and **SD** should be set to 20000 steps/sec^2.

4.5.2 Setting The Speeds And Acceleration/Deceleration In Stepper Modes



The following parameters effect the way moves are executed:

The Creep (Base) Speed is the step rate that all moves will start and finish at, in steps per second. This speed is programmed by the **SC** command, and should be set below the pull in rate of the motor/drive combination, so that the motor will start. It should ideally be set above the resonant frequency range, so that step loss due to resonance will be avoided, unless the minimum speed you wish to use (except Jogs) is below this.

The Slew Speed is set by the **SV** command in steps per second. This is the top speed the motor will travel at during a move. The maximum Slew Speed can be found by trial and error by gradually increasing the speed until a stall occurs, then back it off to a reliable speed. This test should be tried when the motor is at its highest load and maximum operating temperature as the motor torque is reduced with temperature. The initial value for **SV** is 1000 steps per second, which with an HS series stepper motor driven in half step mode equates to 150 rpm.

The rate of change of speeds between the Creep Speed and the Slew Speed is set by the **SA** (set acceleration) command in steps per second per second. Likewise the rate of change of speeds between the Slew Speed and the Creep Speed is set by the **SD** (set deceleration) command in steps per second per second. The ramps are linear. The acceleration setting **SA** and the deceleration setting **SD** can initially be set to give say a 0.5 Sec. ramp time. For example if the value of **SV** is 1000 steps/sec then the values of **SA** and **SD** should be set to 2000 steps/sec^2.

It is important to set the values of **SV, SA** and **SD** safely within the maximum performance attainable by the motor. If the motor stalls, and it has no position sensor, this situation will not be detected.

4.5.3 Setting Creep Steps

The number of Creep Steps is set by the **CR** command in steps. This is the number of steps at the end of a move that will be moved at the Creep Speed (**SC**). It may be important to travel the last part of a move at Creep speed to reduce effects of backlash.

4.5.4 Setting Settling Time

The Settling Time is set by the **SE** command in milliseconds. It is the time that is waited at the end of a move to allow the mechanism to settle, before another move can be started. The default value of 100 ms is suitable for most stepper systems and may be reasonable for servo systems. In high speed servo systems a value of 10 ms might be more appropriate.

4.5.5 Setting the Limit Deceleration

The rate of deceleration when a limit switch is hit is set by the **LD** (limit deceleration) command. In a servo system position will not be lost if a limit switch is activated. However in an open-loop stepper system it might be necessary that position integrity is nor lost if a *hard limit* is activated. The correct setting of the limit deceleration to allow deceleration without step loss can be used to achieve this.

4.5.6 Setting the Soft Limits

The values of upper (**UL**) and lower (**LL**) soft limits can be set to restrict the range of movement of a mechanism. Soft limits are checked when *jogging*, in response to **MA** (move absolute) **MR** (move relative) or **CV** (constant velocity) commands. As the PM600 assumes the position to be zero on power-up, the soft limits are relative to this position. If subsequently the zero position is changed, then the soft limits will also move.
If the Soft Limits are enabled, then any move by **MA** or **MR**, that would end above the Upper Soft Limit or below the Lower Soft Limit will not be executed and would respond with an appropriate error message. It is also not possible to move outside the limit with the manual Jog controls or **CV** (constant velocity) moves.
The Upper Soft Limit is set by the **UL** command and the Lower Soft Limit is set by the **LL** command. The Soft Limits can be disabled (OFF) or enabled (ON) with the Soft Limits (**SL**) command. **SL0** is the command to disable soft limits and **SL1** is the command to enable soft limits. The default setting for the **SL** command is enabled (ON).

4.5.7 Stepper Closed-Loop Functions

In closed-loop stepper control modes (**CM14** and **CM12**) if there is a difference between the Command Position and the Actual Position of greater than the Tracking Window, then the controller will detect that a *tracking error* (stall) has occurred. The Tracking Window is therefore used to set the allowable error before an *tracking error* is detected and is set by the **TR** command in steps. It should be set to a sufficient size to prevent false triggering e.g. by backlash if the encoder is mounted remote from the motor.

If the *tracking abort* is enabled (refer to **AM** command) and a *tracking error* is detected, all moves are stopped and the condition is immediately latched. Any further attempted moves or setting of certain parameters will not be executed and will return a `! TRACKING ABORT` error message. The *tracking error* condition may be reset with the **RS** command, after investigating the cause of the *tracking error*.

At the end of a move (**MA** or **MR**), the difference between the Command Position and the Actual Position is compared with the *Window* (NOT the *Tracking Window*). The *Window* is set by the **WI** command in steps.

If the error between the Command and Actual positions is less than or equal to the Window, then it will continue on to the next command as normal. If it is greater and the PM600 is in control mode 14 (**CM14)** then the controller will start a routine to attempt to correct the error between the Command and Actual positions. The difference between the two is scaled by the *proportional gain* (**KP**) and this value is the amount that will be the size of the correction move for this iteration. If the value is less that 1 step then it will be made 1 step. The move will be made in the appropriate direction at the Creep Speed. If after the correction move, the difference between the Command and Actual positions is still not within the window, then it will attempt another correction move with the current positional error (multiplied by the Correction gain). The controller will therefore repetitively close the error until it is less than the *Window* or a *Time Out* occurs. The correction is only allowed a time defined by the Time Out parameter (set by the **TO** command in milliseconds) to complete the position error correction. If the correction is not within the Window after this time has elapsed, then a *Move Not Complete* condition will be declared. The *proportional gain* is set by the **KP** command in percent and therefore can take a value between 1 and 100. It should be set to avoid over-correction e.g. with a remote encoder and backlash.

4.5.8 Backup

Important – if the set-up of the PM600 is changed by using the above commands, then a **BD** (**B**ackup **D**igiloop parameters) command **must** be issued to save the set-up values to Flash memory. If this not done, the values will be lost on power-down.

4.5.9 Returning To Default Parameters

The initialise (**IN**) command can be used to initialise all the parameters and set-up to the default values. This command can be used if he controller was to be used for a different application.

4.6 Setting the Positions

The command position and the positions for each encoder can be changed to a given value by the following commands:

Set Command Position (**CP**)	Set the Command Position to the given value.
Set Actual Position (**AP**)	Set the Encoder 1 Position to the given value.
Set Auxiliary Position (**TP**)	Set the Encoder 2 Position to the given value.
Set Input Position (**IP**)	Set the Encoder 3 Position to the given value.

4.7 Querying Settings And Positions

It is possible to interrogate the parameter settings and positions of the controller by the following commands:

Output Command Position (**OC**)	returns the current Command Position. This is the position that the controller believes the motor to be at.
Output Actual Position (**OA**)	returns the current Actual Position. This is the position that the motor or mechanism actually is at. The position is read from encoder 1.
Output Input Position (**OI**)	returns the current Input Position. This is the command position that is used both in synchronised moves and 'Input encoder' jogging. The position is read from encoder 3.
Output Auxiliary Position (**OT**)	returns the current Position of the auxiliary encoder. This position is used as a diagnostic tool when setting up the controller. The position is read from encoder 2.
Output Positions (**OD**)	returns the current captured datum position. If no datum position has been captured a **`! NO VALID DATUM`** error message will be returned.
Output Positions (**QP**)	returns a list of the current positions.
Output Velocity (**OV1000**)	returns the current speed of in steps per second. In open-loop stepper mode (**CM11**) the speed is derived from the command position (**CP**). In other modes the speed is derived from the actual position (**AP**).
Output Status (**OS**)	returns a string of '0' or '1' characters to describe the current status of the controller. See programmers reference section.
Output Current Operation (**CO**)	returns the current activity.
Query All (**QA**)	returns a page list of the current positions and set-up parameters.
Query All (**QM**)	returns a list of the current control mode, abort mode, datum mode and jog mode settings.
Query All (**QJ**)	returns a list of the current joystick parameters.

4.8 Move Commands

The motor can be moved to required positions by the **MR** and **MA** commands. Move relative (**MR**) will move the motor by the specified number of steps away from its current position. Move absolute (**MA**) will move the motor to the specified position relative to zero.

While the motor is moving a is shown in the STATUS display. During the deceleration phase after a stop (**ST**) command or an **Escape** character has been received, the STATUS display will show indicating 'Stopping'.
A Constant Velocity (**CV**) command will ramp the motor up to the specified speed and will continue until either instructed to stop with a **ST** command or by hitting a Hard or Soft Limit. While the motor is moving a will be shown in the STATUS display. The speed can be changed during a Constant Velocity move by issuing another **CV** command with a different speed value.

4.9 Manual Jogs

There are three methods of commanding JOG or manual moves:

1. Jog switch inputs using fast (**SF**) and slow jog speeds (**SJ**) such as with a jog box.
2. Joystick input using an analogue voltage such as from an analogue joystick or potentiometer.
3. Input encoder jog using the quadrature signals on the Input encoder such as with a trackerball.

The method can be selected by using the *Jog Mode* (**JM**) command. All three methods can be enabled together but only one is active at a time. The default *Jog Mode* is for the Jog switch inputs only to be active.

During a Jog move the STATUS display will show J .

4.9.1 Jog Switches

Momentarily activating the Jog+ input will step the motor one step in the positive direction. If this input is activated for longer than half a second, the motor will begin to move (in the positive direction) at the programmable Jog Speed defined by the **SJ** (set jog speed) command, for the duration that the input is activated. If the Jog Fast input is activated during this time, the motor will accelerate up to the Fast Jog Speed set by the **SF** (set fast jog speed) command and then decelerate when released.
The same applies to the Jog- input, but the movement will be in the negative direction

4.9.2 Setting Jog Speeds

The Jog Speed is the speed that manual moves are driven when either the Jog+ or Jog- inputs are held active for more than half a second. This is set by the **SJ** command in steps per second. If the Jog Fast input is also active, the speed will ramp up at the **SA** acceleration rate to the Fast Jog Speed set by the **SF** command. If the Jog Fast input is subsequently released the speed will reduce at the **SD** rate until the slow jog speed is reached. The values of **SA** and **SD** are those used in normal moves. The value of **SJ** needs to be found by experimentation. The value of **SF** is typically 5 times the slow jog speed (**SJ**).

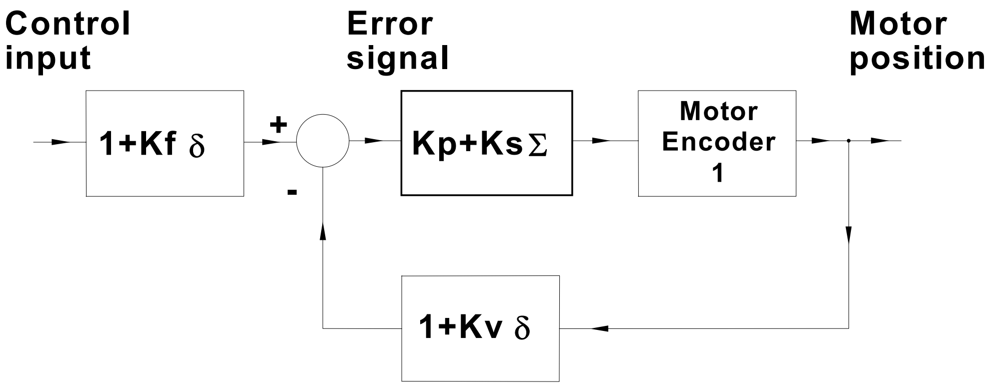
4.9.3 Joystick

Using the Joystick input an analogue voltage such as from an analogue joystick or potentiometer can be a command input. The Joystick must be calibrated using the method described in the Joystick Calibration section. The PM600 uses a non-linear law to determine the required speed from the voltage appearing at the joystick input.

4.9.4 Input Encoder Jog

An incremental encoder can be used to command movement. The quadrature signals from the Input encoder (encoder 3). This input can come from a trackerball or a manual position dial.
The ratio of movement is controlled by the **GR** command.

5 Servo Optimisation



PM600 Single Encoder Coefficient Model

The above diagram illustrates the relationship between the control coefficients in a PM600 servo loop. The *control input* is a number generated by a move command.
The PM600 then generates a signal to drive the motor via a servo amplifier. The encoder (usually mounted on the rear of the motor) produces a feedback signal of the motor's position. This enables the PM600 to calculate a position error signal and continuously update the command signal to the amplifier.

The PM600 in servo mode can be considered as a discrete-time *P.I.D.F.* controller. (Proportional, Integral, Derivative, Feedforward). Coefficients can be varied to change the system characteristics or to optimise the response of the motor in a particular application.

5.1 Optimisation Of System Response.

In a servo control system the following characteristics should be considered in the process of optimisation:

1. Command response - steady-state accuracy, rise time, overshoot, and settling time.
2. Disturbance response - steady-state, transient.
3. Sensitivity to parameter changes.

Some order of priority of the system behaviour should be established to give an objective to the optimisation process. The choice of these parameters is to some extent interactive, and compromises are sometimes necessary in order to achieve a satisfactory solution. The values of **KP**, **KS**, **KV** and **KF** will depend on the characteristics of the system.

KP Proportional gain coefficient.
This coefficient controls the proportional gain in the loop. The controller multiplies the position error by the value of the **KP** coefficient. This influences the amount that the system will react to the error. System stability will limit the maximum value of this coefficient. Increasing the value of **KP** increase the position accuracy and dynamic response.

KV Velocity feedback coefficient.
The value of this coefficient defines the magnitude of the position encoder derived velocity feedback signal. This velocity signal is combined with the position feedback signal, and produces a damping effect.
This coefficient influences the transient response. It has the effect on the system of reducing overshoot and enhancing stability, but too high a value can create a high frequency oscillation (*buzzy* system) and ultimately an unstable system.

KS Sum coefficient.
The controller sums the value of the position error every millisecond. This sum is then multiplied by the coefficient **KS**. A non-zero value of **KS** will result in a final position at the end of a move that has zero error. The larger the value of **KS** the faster the system will reach its final position. Too large a value of **KS** will cause overshoot and system oscillation. The **KV** coefficient is used to damp overshoots and oscillation.

KF Velocity feedforward coefficient.
This coefficient compensates for the servo lag created by the **KV** coefficient.
The value of **KF** should be zero in positioning moves. In a profile or a move where the motor must follow the acceleration/deceleration profile closely, the value of **KF** should be equal to the value of **KV**.

HINTS:

Initially increase **KP** in the sequence 20, 50, 100, 200, 500, etc. This should produce a *stiffer* feel to the motor; try *testing* the motor shaft each time by moving it from its static position and releasing it suddenly. The response created by this method is known as the disturbance response.

Observe the settling behaviour, and when **KP** is at a level that causes motor *ringing* without actual oscillation, begin adding a little **KV**. This should damp out the ringing, but too much will probably cause oscillation. This oscillation will be at a higher frequency than that caused by too much **KP** - a sort of *grittiness*.

Now try adding **KS** to enhance the disturbance response and final position accuracy. **KV** will have to be increased to reduce overshoot.

Remember that an unstable loop might cause damage to a mechanism, so the process should be done with care.

The **IN** (initialise) function will revert the coefficients to default levels of KP=10, KS=0, KV=0 and KF=0.

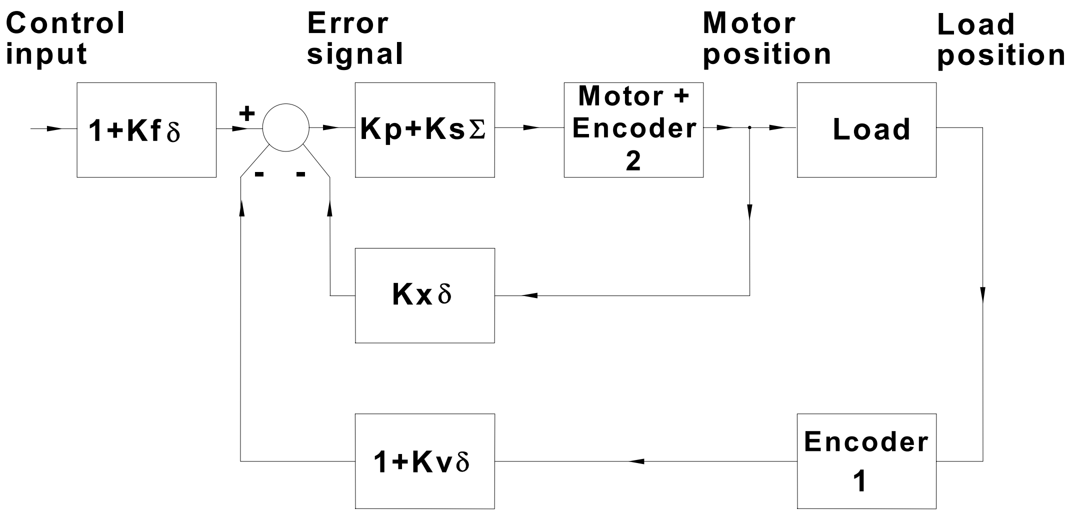
5.1.1 Tune Command

The **TUNE** command can be used to derive an approximate set of servo coefficients. The controller will *exercise* the motor over a small displacement for a few seconds and obtain a set of values for the *K* coefficients that should be stable and provide a reasonable disturbance rejection.
The tuning algorithm can fail if there is excessive backlash, if the low frequency loop gain is either very small or very large or the feedback encoder phasing is wrong. Further optimisation of system response may be required to achieve the desired performance.
If the **TUNE** command fails to find an appropriate set of coefficients a **! TUNE FAILURE** error will be returned.

5.2 Dual Encoder Mode

In high resolution systems where a remote encoder with a large number of counts per revolution of the motor is used, the amount of damping available from the **KV** coefficient can be insufficient. This is a particular problem when there is backlash, compliance and stiction between the motor and the position encoder.

An extra encoder on the rear shaft of the motor can be used to give the required damping factor. The level of the feedback signal from this encoder is controlled by the coefficient **KX**. The position (remote) encoder is connected to *Encoder 1* input and the motor encoder is connected to the *Encoder 2* input.



PM600 Dual Encoder Coefficient Model

The increased resilience of the coupling and the reduction in backlash between Encoder 2 and the motor, compared with that of Encoder 1 and the motor means that the effectiveness of **KX** can be much greater than that of **KV**.

Setting Up

The correct phasing must be established for both encoders. This is dependent on the number of mechanical inversions in the system.

1. If the motor has an associated tacho loop this should be set-up first (set for unity gain).
2. With the motor disconnected from the load, and Encoder 2 disconnected, increase the value of **KX** and disturb the output shaft of the motor. If the motor races off then Encoder 2's phasing must be reversed (Swap A+ & A- with B+ & B-). If it is not possible to get to the motor shaft, then set-up the speed **SV**, acceleration **SA** and deceleration **SD**, and attempt a short move.
3. Connect Encoder 1 and increase the value of **KP**. Attempt a short move. Again if the motor races off then Encoder 1's phasing must be reversed {Swap A+ & A- with B+ & B- or set a negative encoder ratio (**ER**)}.

Monitoring the encoder positions either by using the **QP** (query positions) or using the **OA** (output actual position) and **OT** (output auxiliary position) commands should show that a positive move (encoder 1) gives a positive value of encoder 2 counts.

In practice the process of optimising the coefficients is similar to that of a single encoder system but with **KX** replacing **KV**. Note however that the **TUNE** command only affects **KP, KV, KS and KV** therefore its use in a dual encoder system is inappropriate and will produce a **! TUNE FAILURE** error.

5.3 Optimisation Of Dual Encoder Servo System Response.

As in the single encoder control system the following characteristics should be considered in the process of optimisation:

1. Command response - steady-state accuracy, rise time, overshoot, and settling time.
2. Disturbance response - steady-state, transient.
3. Sensitivity to parameter changes.

Once again some order of priority of the system behaviour should be established to give an objective to the optimisation process. The choice of these parameters is to some extent interactive, and compromises are sometimes necessary in order to achieve a satisfactory solution. The values of **KP**, **KS**, **KV, KF** and **KX** will depend on the characteristics of the system.

The descriptions of **KP** and **KS** are similar to those in single encoder mode.

KX Velocity feedback coefficient.
The value of this coefficient defines the magnitude of the auxiliary encoder derived velocity feedback signal. This velocity signal is combined with the position feedback signal, and produces a damping effect.
This coefficient influences the transient response. It has the effect on the system of reducing overshoot and enhancing stability, but too high a value can create a *buzzy* system, and ultimately an unstable system.
In a dual encoder system **KX** is usually used instead of **KV**, therefore in most systems where **KX** is used the value of **KV** will be zero.

KF Velocity feedforward coefficient.
This coefficient compensates for the servo lag created by the **KV** and **KX** coefficients.
The value of **KF** should be zero in most positioning moves. In a profile or a move where the motor must follow the acceleration/deceleration profile closely, the compensation value of **KF** should cancel the position offset created by **KV** and **KX**. If **KX** is zero then the value of **KF** will be equal to **KV**. When a non-zero value is used for **KX** then the value of **KF** that compensates for the servo lag is dependent on the ratio between encoder 2 and encoder 1.
The ratio between the encoders can be found without taking the mechanical gearing into account, by executing a move, and using the *query positions* (**QP**) command to determine the relative movement.
E.g. if the ratio between the two encoders is 5:1 (encoder 2:encoder 1) then the value of **KF** that compensates exactly for **KX** is 5 times the value of **KX**.

HINTS:

Initially increase **KP** in the sequence 20, 50, 100, 200, 500, etc. This should produce a *stiffer* system. In a dual encoder it is often not possible to move the motor shaft manually. Instead the command response can be tested by executing test moves with high rates of acceleration and deceleration set.

Observe the settling behaviour, and when **KP** is at a level that causes motor *ringing* without actual oscillation, begin adding a little **KX**. This should damp out the ringing, but too much will over-damp the system leading to extended move times or tracking errors.

Now try adding **KS** to enhance the disturbance response and final position accuracy. **KX** may have to be increased to reduce overshoot.

A value of **KF** can be calculated from the ratio between the encoders and the value of **KX**. A value for **KF** that is slightly less than the calculated value is likely to give a good compromise between move time and position overshoot.

Remember that an unstable loop might cause damage to a mechanism, so the process should be done with care.

The **IN** (initialise) function will revert the coefficients to default levels of KP=10, KS=0, KV=0, KX=0 and KF=0.

6 Fault finding.

Here is a list of some common problems with PM600 controllers:

Problem	Possible Cause
Unable to communicate with controller.	All controllers and computer or terminal are not at the same *baud rate* or *word mode*. Incorrect transmit and receive wiring. Handshaking of computer or terminal not disabled. Incorrect commands sent - no axis address or *return* character. RS232 Loop-back terminator not fitted.
Motor moves in the wrong direction.	(Stepper modes) The direction reverse switch on the drive (PM542 & PM546) needs to be turned on. The connections of one phase of the motor need swapping. (Servo mode). Phasing of the position encoder feedback is reversed. Either swap encoder connections A+ and A- with B+ and B-, or use a negative encoder ratio (**ER**) and swap both the motor and the tacho (if used) lead polarities.
Motor races off in one direction (servo mode).	Phasing of the position encoder feedback is reversed. In this case either swap encoder connections A+ and A- with B+ and B-, or use a negative encoder ratio (**ER**).
Motor does not move.	**K** coefficients not set - system initialised. Motor stalled. Controller aborted. Hard limits activated. Abort Stop activated.
Motor Stalled.	No encoder feedback - encoder connected to the wrong input. Encoder type switch SW4 is set incorrectly. Incorrect motor wiring. Insufficient motor or drive current.
Tune failure.	Encoder connections incorrect. Servo amplifier settings incorrect - motor tacho reversed (if used) amplifier not set for *flat* gain, current limit too low. Unsuitable load - offset or high inertia, backlash. Large gear ratio between motor and position encoder.
Unstable motor.	Incorrect **K** coefficients Servo amplifier settings incorrect - motor tacho reversed (if used) amplifier not set for *flat* gain, current limit too low.
Position seems to drift.	Noise on encoder signals - unsuitable cabling. Encoder fixing failure - loose grub screws.

7 Programmers Reference

GETTING STARTED COMMANDS			
HC	***H***elp with ***C***ontrol modes	HE	***HE***lp pages
HM	***H***elp with ***M***odes	**HN**	Display ***N***ext Help Page
HP	Display ***P***revious Help Page	**HS**	***H***elp with ***S***tatus Output (**OS**)
HP	Display ***P***revious Help Page	**IN**	***IN***itialise
TUNE	Auto ***TUNE***	**QA**	***Q***uery ***A***ll
QK	***Q***uery constants (***K***)	**QS**	***Q***uery ***S***peeds
ABORT, STOP & RESET COMMANDS			
CONTROL **C**	Hard Stop	**ESC**	Soft Stop
AM<mode>	Set ***A***bort ***M***ode	**AB**	Command ***AB***ort
RS	***R***e***S***et	**QM**	**Q**uery **M**ode
ST	Soft ***ST***op		
INFORMATION			
CO	Display the ***C***urrent ***O***peration	**ID**	***ID***entify Version
OC	***O***utput ***C***ommand position	**OA**	***O***utput ***A***ctual position (Encoder 1)
OT	***O***utput Auxiliary Position (Encoder 2)	**OI**	***O***utput ***I***nput position (Encoder 3)
OD	***O***utput ***D***atum position	**OV**	***O***utput ***V***elocity
OS	***O***utput ***S***tatus string	**OF**	***O***utput ***F***ollowing Error
QA	***Q***uery ***A***ll	**QK**	***Q***uery constants (***K***)
QS	***Q***uery ***S***peeds	**QP**	***Q***uery ***P***ositions
QM	***Q***uery ***M***odes	**QL**	***Q***uery Privilege ***L***evel
SET UP			
CM<mode>	Set ***C***ommand ***M***ode	**ER**<numerator>/<denominator>	Set ***E***ncoder ***R***atio
BO<steps>	Set ***B***ack***O***ff Steps	**CR**<steps>	Set ***Cr***eep steps
TO<value>	Set ***T***ime***O***ut	**SE**<time>	Set ***SE***ttling time
WI<steps>	Set settling ***W***indow		
FAULT DETECTION FEATURES			
SL<mode>	Set ***S***oft ***L***imits	**TH**<value>	Set Motor Stalled ***Th***reshold
TR<value>	Set ***TR***acking window		
		TO<value>	Set ***T***ime***O***ut
SERVO COEFFICIENTS			
KF<value>	Set ***F***eedforward coefficient	**KP**<value>	Set ***P***roportional gain coefficient
KS<value>	Set ***S***um coefficient	**KV**<value>	Set ***V***elocity damping coefficient
KX<value>	Set e***X***tra velocity feedback coefficient	**QK**	***Q***uery constants (***K***)
DATUMING			
CD	***C***lear Captured ***D***atum Position	**OD**	***O***utput ***D***atum position
HD<direction>	Go ***H***ome to ***D***atum	**MD**	***M***ove to ***D***atum Position
SH<position>	***S***et ***H***ome Position	**DM**<mode>	Set ***D***atum ***M***ode
QM	***Q***uery ***M***odes		
POSITION COMMANDS			
AP<position>	Set ***A***ctual ***P***osition	**CP**<value>	Set ***C***ommand ***P***osition
IP<position>	Set ***I***nput encoder's ***P***osition	**TP**<position>	Set Auxiliary ***P***osition
DA<position>	***D***ifference ***A***ctual position	**DI**<position>	***D***ifference ***I***nput encoder's position

SPEED, ACCELERATION AND DECELERATION			
CV<velocity>	***C***onstant ***V***elocity mode	**SC**<speed>	***S***et ***C***reep speed
SF<speed>	***S***et ***F***ast jog speed	**SJ**<speed>	***S***et slow ***J***og speed
SV<speed>	***S***et ***V***elocity	**SA**<acceleration>	***S***et ***A***cceleration
SD<deceleration>	***S***et ***D***eceleration	**LD**<deceleration>	Set ***L***imit ***D***eceleration
MOVES			
BO<steps>	Set ***B***ack***O***ff Steps	**CR**<steps>	Set ***Cr***eep steps
MA<position>	***M***ove ***A***bsolute	**MR**<position>	***M***ove ***R***elative
GM<steps>	***G***earbox Offset ***M***ove	**HD**<direction>	Go ***H***ome to ***D***atum
MD	***M***ove to ***D***atum Position	**DE**<time>	Set ***DE***lay time
SOFT LIMITS			
LL<position>	Set ***L***ower *soft* ***L***imit	**UL**<position>	Set ***U***pper *soft* ***L***imit
SL<mode>	Set ***S***oft ***L***imits		
GEARBOX			
GA	***G***earbox ***A***bsolute mode	**GB**	***G***ear***B***ox mode
GR<numerator>/<denominator>	***G***earbox ***R***atio	**GM**<steps>	***G***earbox Offset ***M***ove
GD<value>	Set ***G***earbox ***D***enominator	**GN**<value>	Set ***G***earbox ***N***umerator
WS	***W***ait for ***S***ynchronisation		
END OF MOVE			
SE<steps>	Set ***SE***ttling time	**TO**<time>	Set ***T***ime***O***ut
WI<steps>	Set settling ***Wi***ndow	**WE**	***W***ait for ***E***nd of current move
READ & WRITE PORTS			
RP	***R***ead ***P***ort	**WP**<bit pattern>	***W***rite ***P***ort
WA<bit pattern>	***WA***it for input event		
JOG			
JM<mode>	Set ***J***og ***M***ode	**SF**<speed>	***S***et ***F***ast jog speed
SJ<speed>	***S***et slow ***J***og speed	**JC**<value>	Set ***J***oystick ***C***entre Position
JR<value>	Set ***J***oystick ***R***ange	**JS**<speed>	Set ***J***oystick ***S***peed
JT<value>	Set ***J***oystick ***T***hreshold	**QJ**	***Q***uery ***J***oystick Settings
ANALOGUE INPUT AND OUTPUTS			
AI<channel>	Query ***A***nalogue ***I***nput	**AO**<channel/value>	Set ***A***nalogue ***O***utput
AL<channel/value>	Wait for ***A***nalogue ***L***ess than Value	**AG**<channel/value>	Wait for ***A***nalogue ***G***reater than Value
SEQUENCES			
AE<sequence no.>	***A***uto-***E***xecute sequence	**AD**	***A***uto-Execute ***D***isable
DS<sequence no.>	***D***efine ***S***equence	**ES**	***E***nd ***S***equence definition
LS<sequence no.>	***L***ist ***S***equence	**XS**<sequence no.>	E***X***ecute ***S***equence
BS	***BACKUP SEQUENCE***	**US**<sequence no.>	***U***ndefine ***S***equence
IF	Do next command ***I***f ***F***alse	**IT**	Do next command ***I***f ***T***rue

PROFILES			
DP<profile no.>	***D***efine ***P***rofile	**EP**	***E***nd ***P***rofile definition
LP<profile no.>	***L***ist ***P***rofile	**XP**<profile no.>	E***X***ecute ***P***rofile
BP	***B***ackup ***P***rofiles	**UP**<profile no.>	***U***ndefine ***P***rofile
PT<value>	***P***rofile ***T***ime		
CAMS			
BC	***B***ackup ***C***ams	**DC**<cam no.>	***D***efine ***C***am
EC	***E***nd ***C***am definition	**LC**<cam no.>	***L***ist ***C***am
XC<cam no.>	E***X***ecute ***C***am	**XR**<cam no.>	E***X***ecute Cam ***R***elative
UC<cam no.>	***U***ndefine ***C***am	**XY**<x position>/<y position>	Cam co-ordinates
PRIVELEGE LEVEL			
NP<new PIN>	***N***ew ***P***in	**PI**	Enter ***PI***N
PL	Set ***P***rivilege ***L***evel	**QL**	***Q***uery Privilege ***L***evel
HELP			
HE	Display ***HE***lp Pages	**HN**	Display ***N***ext Page
HP	Display ***P***revious Page	**HM**	Display ***H***elp with ***M***odes Commands
BACKUP			
BA	***B***ackup ***A***ll	**BC**	***B***ackup ***C***ams
BD	***B***ackup ***D***igiloop Parameters	**BP**	***B***ackup ***P***rofiles
BS	***B***ackup ***S***equence		

CONTROL **C** (ASCII *03*) **Hard Stop**.

Operates on all axes.
Moves, sequences and profiles halted immediately.
Motion stopped at the **LD** rate.
Command buffer cleared.
Sets status to **Idle**.
Sequences and profiles retained in memory.

ESCAPE (ASCII 27) **Soft Stop**

Operates on all axes.
Moves, sequences and profiles halted immediately.
Motion stopped at the **SD** rate.
Command buffer cleared.
Status returns to **Idle**.
Sequences and profiles retained in memory.

7.1 PM600 Series Commands

AB ABORT

The control of the motor can be aborted by sending **AB**. When aborted, the servo loop is disabled and the status display will show **C** for command abort. A user abort can be reset with the **RS** command. The encoder positions are still read while aborted.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>AB	N/A	N/A		N/A	0

Condition Requirements
None.

Notes:
The response to a CO command will be Command Abort.
It will override any other abort situation.

Responses:

! COMMAND ABORT	Command has been accepted.

Example:

1AB	Abort axis 1.

AD AUTO-EXECUTE SEQUENCE - DISABLE

Switches off any auto-execute sequences that may have been set by the **AE** command. Note that this setting is written to the non volatile FLASH memory and is therefore retained after power down.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>AD				Disabled	5

Condition Requirements
Idle

Notes:
Value stored in FLASH

Responses:

OK	Command has been accepted.

Example:

1AD	Disable auto execute of axis 1.

AE AUTO-EXECUTE SEQUENCE - ENABLE

Set sequence *n* to run on power-up of the controller (auto-execute). This can be used in stand alone systems where there is no permanent host computer or terminal. Note that this setting is written to the non volatile FLASH memory and is therefore retained after power down.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>AEn	Seq. No.	0	7	Disabled	5

Condition Requirements
Idle

Notes:
Value stored in FLASH

Responses:

OK	Command has been accepted.
! INVALID SEQUENCE NUMBER	Argument is out of valid range.
! SEQUENCE UNDEFINED	Sequence specified has not been defined yet.

Example:

1AE5	Sets auto execute of axis 1 to run sequence 5 on power-up.

AG WAIT FOR ANALOGUE INPUT GREATER THAN

Wait until an Analogue Input is greater than a value. The first argument is the analogue input number and the second argument is the value that the analogue input is compared with. If the first argument is 1 or 2 then the value will be read from the bi-polar Analogue inputs 1 or 2 respectively. The value is between –2047 and +2047 at a scaling of 5mV per unit (±10V). If the first argument is 3 to 5, then the unipolar values are for Joystick1, Joystick2 and JoystickCT respectively for the range 0 to 4095 for 0V to 5V.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>AGn/xxx					1
n	N/A	1	5	N/A	
N = 1 or 2 xxx	N/A	-2047	+2047	N/A	
N= 3 to 5 xxx	N/A	0	4095	N/A	

Condition Requirements **Notes:**
None

Response:
OK The condition has come true or
! NOT VALID ADC CHANNEL

Example:

1AG2/1000	Wait until Analogue Input 2 is greater than 5V
1AG3/2047	Wait until Joystick 1 Input is greater than 2.5V

AI READ ANALOGUE INPUT

Read Analogue Input. If the argument is 1 or 2 then the value will be read from the bi-polar Analogue inputs 1 or 2 respectively. These return a value between –2047 and +2047 at a scaling of 5mV per unit (±10V). For arguments 3 to 5 will give the unipolar values for Joystick1, Joystick2 and JoystickCT respectively for the range 0 to 4095 for 0V to 5V.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>AIn	N/A	1	5	N/A	0

Condition Requirements **Notes:**
None

Response:
The response is a string of numeric characters. If the communications are in Verbose Mode, the reply is preceded by something of the format **ADC1 =** or **! NOT VALID ADC CHANNEL**

Example:
If the controller of axis 1 currently has a Joystick 1 signal of about 2.5V then the command:

1AI3	in Verbose Mode will respond:	**01:Joystick 1 = 2040**
1AI3	in Quiet Mode will respond:	**01:2040**

AL WAIT FOR ANALOGUE INPUT LESS THAN

Wait until an Analogue Input is less than a value. The first argument is the analogue input number and the second argument is the value that the analogue input is compared with. If the first argument is 1 or 2 then the value will be read from the bi-polar Analogue inputs 1 or 2 respectively. The value is between –2047 and +2047 at a scaling of 5mV per unit (±10V). If the first argument is 3 to 5, then the unipolar values are for Joystick1, Joystick2 and JoystickCT respectively for the range 0 to 4095 for 0V to 5V.

Syntax	Units	Range	To	Initial State	Privilege level
<ad>ALn/xxx					1
n	N/A	1	5	N/A	
N = 1 or 2 xxx	N/A	-2047	+2047	N/A	
N= 3 to 5 xxx	N/A	0	4095	N/A	

Condition Requirements
None

Notes:

Response:

OK	The condition has come true or
! NOT VALID ADC CHANNEL	

Example:

1AL2/-500	Wait until Analogue Input 2 is less than 2.5V
1AL3/2047	Wait until Joystick 1 Input is less than 2.5V

AM SET ABORT MODE

Set the conditions that cause an abort and disable the control (servo) loop.

Syntax	Units	Range	To	Initial State	Privilege level
<ad>AMabcdefgh	Bits	0	1	00000000	6

where **a** - 0 – Abort Stop Input disables control loop
1 – Abort Stop Input stops all moves only
b - 0 –Abort Stop Input is latched requiring RS command to reset
1 – Abort Stop Input is only momentary
c - 0 – Stall Error disables control loop
1 – Stall Error is not detected
d - 0 – Tracking Error disables control loop
1 – Tracking Error is indicated but control loop remains active
e - 0 – TimeOut Error disables control loop
1 – TimeOut Error finishes move and control loop remains active
f - Reserved for future use.
g - Reserved for future use.
h - 0 – Enable output switched OFF during a disabled control loop
1 – Enable output left ON during a control loop abort

Condition Requirements
Idle

Notes:
Bit **a** = **0** and bit **b** = **1** is not appropriate and should not be used.

Response:

OK	Command has been accepted.
! OUT OF RANGE	Illegal argument range or format

Examples:

1AM00010001	Set axis 1 to abort on all conditions except Tracking Error, enable output stays ON.
1AM11000000	Set axis 1 to abort on all conditions except momentary Abort Stop input only stops moves.

Caution: Do not disable fault detection features unless sure of operation.

AO SET ANALOGUE OUTPUT

Set the voltage of an Analogue Output. The first argument is 1 or 2 and the value will be written to the bi-polar Analogue outputs 1 or 2 respectively. The value is between –2047 and +2047 at a scaling of 5mV per unit (±10V).
In servo mode, Analogue Output 1 is used for the velocity command signal that is connected to the servo amplifier. So only AO2/xxx can be used in servo mode, but both can be used in stepper motor modes.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>AOn/xxx					3
n	N/A	1	2	N/A	
xxx	N/A	-2047	+2047	N/A	

Condition Requirements
None

Notes:

Response:

OK	Command has been accepted or
! NOT VALID DAC CHANNEL	

Examples:

1AO1/1024	Set the Analogue output 1 to +5V
1AO2/-2047	Set the Analogue output 2 to -10V

AP SET ACTUAL POSITION

Set the actual position (position of motor or mechanism) value to that given in the argument.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>APnnn	Steps	-2147483647	2147483647(±2^{32})	0	3

Condition Requirements
Idle, Constant velocity or Gearbox

Notes:
Value zero on power-up.

Response:

OK	Command has been accepted.

Examples:

	1AP5000	Set the axis 1 Position to 5000.
or	1AP0	Set the axis 1 Position to zero.

BA BACKUP ALL

Saves parameters, sequence definitions, profile definitions and cam definitions to non-volatile *flash* memory. These are then restored on power-up.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>BA	N/A	N/A		N/A	5

Condition Requirements
Idle

Notes:

Response:

OK	Command has been accepted.

BC BACKUP CAMS

Saves all cam definitions to non-volatile *flash* memory. These are then restored on power-up.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>BC	N/A	N/A		N/A	5

Condition Requirements
Servo mode, Idle

Notes:

Response:

OK	Command has been accepted.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

BD BACKUP DIGILOOP PARAMETERS

Saves Digiloop parameters to non-volatile *flash* memory. These are then restored on power-up.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>BD	N/A	N/A		N/A	4

Condition Requirements
Idle

Notes:

Response:

OK	Command has been accepted.

BO SET BACKOFF

Set number of back-off steps that are executed at the end of a move. The motor will decelerate to the creep speed at the back-off position relative to the required end position. The controller will then complete the move at the creep speed. The controller will therefore always approach the final position at the creep speed and from the same direction. This may be useful in combating backlash in a mechanism.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>BOnnn	Steps	-32000	32000	0	6

Condition Requirements
Idle

Notes:
Value stored in FLASH by BD command.

Response:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Examples:

1BO500	Set the back-off distance to 500 steps on axis 1.

BP BACKUP PROFILES

Saves all profile definitions to non-volatile *flash* memory. These are then restored on power-up.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>BP	N/A	N/A		N/A	5

Condition Requirements
Servo mode, Idle

Notes:

Response:

OK	Command has been accepted.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

BS BACKUP SEQUENCES

Saves all sequence definitions to non-volatile *flash* memory. These are then restored on power-up.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>BS	N/A	N/A		N/A	5

Condition Requirements
Idle

Notes:

Response:

OK	Command has been accepted.

CD CLEAR CAPTURED DATUM

Clears the current captured datum position.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>CD	N/A	N/A		N/A	3

Condition Requirements
None

Notes:

Response:

OK	Command has been accepted.

CH SET ACTIVE CHANNEL

Sets the active channel. Parameters such as positions and speeds are changed to those relating to the selected channel. The *step* and *direction* signals are routed to the outputs associated with the selected channel. The setting of the channel number also selects the correct *limit* and *datum* inputs.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>CHn	N/A	1	NC setting	1	1

Condition Requirements
PM608 Only

Notes:
Active channel number is shown on front panel display.

Response:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

CL CLONE CURRENT SETTINGS

Copy current settings to another channel.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>CLn	N/A	1	8	N/A	6

Condition Requirements
PM608 Only

Notes:
Channel to be copied to is not constrained by the setting of the number of channels.
To store the settings in FLASH memory use a **BD** command.

Response:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

CM SET CONTROL MODE

Sets the current control mode. When changing from servo mode to a stepper mode or vice versa, the controller will be forced into a *command abort* and will reset all parameters to the default values for the new mode. The *command abort* can be reset with an **RS** (reset) command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>CMnn	N/A	1, 11, 12, 13 or 14	N/A	1	8

CM1	Servo mode
CM11	Open loop stepper mode
CM12	Checking stepper mode
CM13	External loop stepper mode
CM14	Closed loop stepper mode

Condition Requirements
Idle

Notes:
Forces a *Command Abort* if changing between servo/stepper
PM608 Auto-detects mode and sets to CM11

Response:

OK	Command has been accepted (no servo/stepper change).
! COMMAND ABORT	Command has been accepted (servo/stepper change forced abort).

CO QUERY CURRENT OPERATION

This command will return the current operation being executed by the controller.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>CO	N/A	N/A		N/A	0

Condition Requirements
None

Notes:

Responses:

Mode = and one of the following:

Backoff	Executing backoff correction
Cam	Executing Cam profile
Cam synchronisation	Waiting for synchronisation in cam mode
Command Abort	Aborted due to command abort (AB) command
Constant velocity	Constant velocity
Correcting	Closed loop stepper correcting
Creep	Creep steps at end of move
Delay	Executing delay command
Gearbox	Gearbox mode
Gearbox synchronisation	Waiting for synchronisation in gearbox absolute mode
Home to datum	Searching for datum
Idle	No moves or waits taking place and no aborts have occurred
Input abort	Aborted due to Abort Stop input
Jogging	Jogging or Joystick move
Move	Move (MA or MR command)
Not Complete /Timeout Abort	Aborted due to Timeout Abort
Profile	Executing profile
RS232 abort	Illegal serial character(s) received
Settling	Settling at end of move
Stopping	Decelerating due to limit, Ctrl-C or ESCAPE command
Stall Abort	Aborted due to Stall Error
Tracking Abort	Aborted due to Tracking Abort
Wait for condition	Waiting for specific pattern on read port

CP SET COMMAND POSITION

Set the command position value to that given in the argument. The command position is the position generated by a move command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>CPnnn	Steps	-2147483647	2147483647(±2^{32})	N/A	3

Condition Requirements
Idle, Constant velocity or Gearbox

Notes:
Value zero on power-up.

Response:

OK	Command has been accepted.

Examples:

	1CP5000	Set the axis 1 Command Position to 5000.
or	**1CP0**	Set the axis 1 Command Position to zero.

CR SET CREEP DISTANCE

Set number of creep steps at the end of a move. The motor will decelerate and execute this number of steps at the creep speed.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>CRnnn	Steps	0	2147483647(2^{32})	0(servo) 10(stepper)	6

Condition Requirements	Notes:
Idle	Value stored in FLASH by BD command.

Response:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Examples:

1CR50	Set the creep distance to 50 steps on axis 1.

CV CONSTANT VELOCITY MOVE

A Constant velocity move is used to move continuously at the required speed. Initially the move will ramp up to the speed given in the argument, where the sign dictates the direction of movement. The argument therefore controls the velocity. Subsequent CV commands can then be sent to change the required velocity, including changes in direction.
The speed is changeable whilst motion is in progress. The **SA** and **SD** rates define the rate at which the change of speed will be made. An ST command, ESCAPE or Control C exits constant velocity mode.
The soft limits are active in CV mode. For continuous applications they must be disabled with the SL command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>CVnnn	steps/sec	-1,200,000	1,200,000	N/A	1

Condition Requirements	Notes:
Idle or Constant velocity.	

Responses:

OK	Command has been accepted.
! HARD LIMIT	Hard limit for required direction is already activated
! SOFT LIMIT	Soft limit for required direction has already been reached
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A timeout abort has been detected

Examples:

1CV2000	Start constant velocity move in positive direction at 2000 steps/sec on axis 1.
1CV-10000	Start constant velocity move in negative direction at 10,000 steps/sec on axis 1.

DA DIFFERENCE ACTUAL POSITION

Add value to actual (position encoder's) position. The command position **CP** is also adjusted by the same amount.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>DAnnn	Steps	-2147483648	2147483647(± 2^{32})	N/A	3

Condition Requirements
None.

Notes:

Response:

OK	Command has been accepted.

Examples:

```
1OA         Get the axis 1 Actual Position.
Response:   15000

1DA5000     Difference axis 1 actual position by 5000.
1OA         Get the axis 1 Actual Position.
Response:   20000
```

DC DEFINE CAM

This command will start a Cam profile definition. There are eight cams that can be defined and the argument selects which profile is to be defined (0 to 7). The only command that is used during a Cam profile definition is **XY.** Any other commands except for **EC** will cause a **!ILLEGAL CAM INSTRUCTION** error.
The commands that follow this **DC** command will not be executed, but will be stored in the on board volatile memory until the End Cam definition (**EC**) command is received. If a Control-C or ESCAPE command is received or the controller runs out of memory, the Cam definition will cease, the Cam will not be stored and the controller will return to the idle state. The Cam Modulo (profile length) is calculated automatically.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>DCn	Cam number	0	7	N/A	5

Condition Requirements
Servo mode, Idle

Notes:
If a Cam is defined, that fact is shown on the QA page.
Use BC command to store cams in non-volatile flash memory.

Responses:

OK: START OF CAM	Command has been accepted.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

```
1DC3          Start definition of Cam 3.
1XY200/500    Second Cam profile point. (First Cam profile co-ordinates 0,0.)
1XY400/500    Next Cam Profile point.
1XY600/-200          "
1XY700/-200          "
1XY750/0             "
1EC           End of Cam profile definition.
```

DE DELAY

This command will start a delay timer for the length given in the argument. After the expiry of this time, the controller will return to the idle mode.
If the following command should not be executed until the end of this delay time, and does not wait for the idle state itself, then you must follow it with a Wait for End (**WE**) command. This will wait until the controller returns to the idle mode and will therefore sense the expiry of the delay time.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>DEnnn	Milliseconds	1	2,000,000	N/A	1

Condition Requirements **Notes:**
Idle

Responses:
OK Command has been accepted.

Example:

1DE5000	Delay for 5 seconds
1WE	Wait for end of delay
1WP22222221	Put write port 1 on after delay.

DI DIFFERENCE INPUT POSITION

Add value to Input position.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>DAnnn	Steps	-2147483648	2147483647(± 2^{32})	N/A	3

Condition Requirements **Notes:**
Idle or Constant velocity.

Response:
OK Command has been accepted.

Examples:

1OI	Get the axis 1 Input Position.
Response:	15000
1DI5000	Difference axis 1 Input position by 5000.
1OI	Get the axis 1 Input Position.
Response:	20000

DM SET DATUM MODE

Set the mode of operation for datum searches.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>DMabcdefgh	Bits	0	1	00000000	6

where **a** - 0 – Encoder index input polarity is normal
1 – Encoder index input polarity is inverted
b - 0 –Datum point is captured only once (i.e. after HD command)
1 – Datum point is captured each time it happens
c - 0 – Datum position is captured but not changed
1 – Datum position is set to Home Position (**SH**) after datum search (**HD**)
d - 0 – Automatic direction search disabled
1 – Automatic direction search enabled
e - 0 – Automatic opposite limit search disabled
1 – Automatic opposite limit search enabled
f - Reserved for future use.
g - Reserved for future use.
h - Reserved for future use.

Condition Requirements
Idle

Notes:

Response:

OK	Command has been accepted.
! OUT OF RANGE	Illegal argument range or format

Examples:
1DM00100000 Set axis 1 to normal datum capture, with automatic setting of the captured position to Home position.

DP DEFINE PROFILE

This command will start a Profile definition. There are eight profiles that can be defined and the argument selects which sequence is to be defined (0 to 7). The only command that is used during a Profile definition is **MR** any other commands except for **EP** will cause a **!ILLEGAL PROFILE INSTRUCTION** error.
The commands that follow this **DP** command will not be executed, but will be stored in the on board volatile memory until the End Profile definition (**EP**) command is received. If a Control-C or ESCAPE command is received or the controller runs out of memory, the Profile definition will cease, the Profile will not be stored and the controller will return to the idle state.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>DPn	Profile No.	0	7	N/A	5

Condition Requirements
Servo mode, Idle.

Notes:
If a Profile is defined, that fact is shown on the QA page.
Use **BP** command to store profiles in non-volatile flash memory.

Responses:

OK	Command has been accepted.
!ILLEGAL PROFILE INSTRUCTION	Command has not been accepted
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

1DP	Start Profile definition.
1MR200	First Profile move.
1MR500	Next Profile move.
1MR-500	“
1MR-200	“
1MR50	“
1EP	End of Profile definition.

DS DEFINE SEQUENCE

This command will start a sequence definition. There are eight sequences that can be defined and the argument selects which sequence is to be defined (0 to 7). All valid commands that follow this **DS** command will not be executed, but will be stored in the on board volatile memory until the End Sequence definition (**ES**) command is received. If a command is not suitable for inclusion in a sequence, the controller will respond **!ILLEGAL SEQUENCE INSTRUCTION**.
If a Control-C or ESCAPE command is received or the controller runs out of memory, the sequence definition will cease, the sequence will not be stored and the controller will return to the idle state. . Note that a **BS** command will be needed to copy the sequences to the non volatile FLASH memory, otherwise it will be lost on power down.

Syntax	**Units**	**Range**	**to**	**Initial State**	**Privilege level**
<ad>DSn	Seq. No.	0	7	N/A	5

Condition Requirements
Idle

RAM

Notes:
The sequences defined are shown on the QA page.
Use **BS** command to store sequences in non-volatile flash

Responses

OK: START OF SEQUENCE	Command has been accepted.

Example:

1DS4	Start definition of sequence 4.
1SV2000	Set slew speed.
1MA8000	First move (absolute).
1MR5000	Next move (relative).
1MR3000	Next move (relative).
1SV50000	Set new slew speed.
1MA0	Next move (return to start position).
1XS4	Execute sequence 4 (loop to start of this sequence).
1ES	End of sequence definition.

EC END CAM DEFINITION

This command will end a Cam profile definition. The Cam definition must have been started by the Define Cam (**DC**) command. No argument is necessary as the sequence number is specified with the Define Cam (**DC**) command.

Syntax	**Units**	**Range**	**to**	**Initial State**	**Privilege level**
<ad>EC	N/A	N/A		N/A	5

Condition Requirements
Servo mode, Define Cam

Notes:

Responses

OK	Command has been accepted.
! ILLEGAL INSTRUCTION	EC attempted when NOT already defining a Cam.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

1DC	Start definition of Cam.
1XY200/500	Second Cam profile point.
1XY400/500	Next Cam Profile point.
1EC	End of Cam definition.

EP END PROFILE DEFINITION

This command will end a Profile definition. The Profile definition must have been started by the Define Profile (**DP**) command. No argument is necessary as the sequence number is specified with the Define Profile (**DP**) command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>EP	N/A	N/A		N/A	5

Condition Requirements
Servo mode, Define Profile

Notes:

Responses:

OK	Command has been accepted.
! ILLEGAL INSTRUCTION	EP attempted when NOT already defining a Profile.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

1DP	Start Profile definition.
1MR200	First Profile move.
1MR500	Next Profile move.
1EP	End of Profile definition.

ER ENCODER RATIO

Set encoder ratio. The encoder ratio is specified by two arguments separated by a **/** character, and is therefore specified as a fraction with the format: numerator**/**denominator. The incoming position encoder pulses and then scaled by this ratio to derive the Actual Position.
Great care must be exercised in setting this ratio as it will affect the stability of a servo loop (K constants etc.). Beware that if you scale the position up (fraction of greater than 1), then certain positions will then become unobtainable.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>ERnnn/nnn					
Numerator	N/A	-32768	32767 ($\pm 2^{15}$)	1	7
Denominator	N/A	1	32767 (2^{15})	1	7

Condition Requirements
Idle.

Notes:
Value stored in FLASH by BD command.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Examples:

1ER400/2000 (1ER1/5) Axis 1 Set encoder gearbox ratio to 1:5 - i.e. for every 5 steps of the position encoder the Actual Position will change by 1 steps. This can be used if for example a 400 step/rev motor is fitted with a 2000step/rev encoder.
1ER-1/1 Axis 1 Set encoder gearbox ratio to -1:1 - i.e. reverse the direction sense of the encoder.

ES END SEQUENCE DEFINITION

This command will end a sequence definition. The sequence definition must have been started by the Define Sequence (**DS**) command. No argument is necessary as the sequence number is specified with the Define Sequence (**DS**) command.

Syntax	**Units**	**Range**	**to**	**Initial State**	**Privilege level**
<ad>ES	N/A	N/A		N/A	5

Condition Requirements
Define Sequence

Notes:

Responses

OK	Command has been accepted.
! ILLEGAL INSTRUCTION	ES attempted when NOT already defining a sequence.

Example:

1DS2	Start definition of sequence 2.
1MR400	First move (relative).
1MR-400	Next move (relative).
1XS5	Execute sequence 5 (transfer control to start of this sequence 5).
1ES	End of sequence definition.

GA GEARBOX ABSOLUTE MODE

This command will enter gearbox mode when the value of the Input (master) encoder is equal to the value of the Actual Position (slave) encoder. The slave motor will then be driven at a ratio of the Input encoder speed. The ratio is specified by the gear ratio command **GR**. Gearbox mode is exited by a **ST** command, **ESCAPE** or Control **C**.
The response to a **CO** command while synchronising will be **01:Gearbox synchronisation** until the *input position* is equal to the *actual position*. After this the response will be the same as in standard (relative) gearbox mode i.e. **01:Gearbox**.

Syntax	**Units**	**Range**	**to**	**Initial State**	**Privilege level**
<ad>GA	N/A	N/A		N/A	1

Condition Requirements
Servo mode, Idle.

Notes:
In absolute gearbox mode SA, SD and SV values are not active.

Responses:

OK	Command has been accepted.
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A timeout abort has been detected
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

1GA	Axis 1 enter absolute gearbox ratio mode.

GB GEARBOX MODE

Enter gearbox mode. The slave motor is now driven at a ratio of the Input encoder speed. The ratio is specified by the gear ratio command **GR**. Gearbox mode is exited by an **ST** command, **ESCAPE** or Control **C**.
The response to a **CO** command while in gearbox mode will be **01:Gearbox**.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>GB	N/A	N/A		N/A	1

Condition Requirements
Servo mode, Idle.

Notes:
In gearbox mode, SA, SD and SV values are not active.

Responses:

OK	Command has been accepted.
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A timeout abort has been detected
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

1GB Axis 1 enter gearbox ratio mode.

GD GEARBOX RATIO DENOMINATOR

Set gearbox ratio denominator. This command can be used in conjunction with GN gearbox numerator. The ratio is therefore specified as a fraction with the format: numerator(**GN**)/(**GD**)denominator. The **GR** gearbox ratio command can also be used to set the ratio.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>GDnnn	N/A	1	32767 (2^{15})	1	4

Condition Requirements
Idle or Gearbox.

Notes:
Value stored in FLASH by BD command.
GR is Not available in sequence definition – use GN & GD.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1GN2 1GD5 Axis 1 Set electronic gearbox ratio to 2:5 - i.e. for every 5 steps of the input encoder the command position will change by 2 steps.

GM GEARBOX MOVE RELATIVE

When in gearbox mode, this command can be used to superimpose a relative move on top of the gearbox slaving. In this way, a correction in the synchronism of the two positions can be changed without exiting the gearbox mode. This move is done at the creep speed.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>GMnnn	Steps	-2147483647	2147483647(±2^{32})	0	1

Condition Requirements
Servo mode, Gearbox mode.

Notes:

Responses:

OK	Command has been accepted.
! NOT ALLOWED IN THIS MODE	Not in Gearbox mode
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A timeout abort has been detected
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

1GM100 Superimpose a move of 100 steps (positive) on top of gearbox ratio mode.

GN GEARBOX RATIO NUMERATOR

Set gearbox ratio numerator. This command can be used in conjunction with GD gearbox denominator. The ratio is therefore specified as a fraction with the format: numerator(**GN**)/(**GD**)denominator. The **GR** gearbox ratio command can also be used to set the ratio.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>GNnnn	N/A	-32768	32767 (±2^{15})	1	4

Condition Requirements
Idle or Gearbox.

Notes:
Value stored in FLASH by BD command.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1GN2 1GD5 Axis 1 Set electronic gearbox ratio to 2:5 - i.e. for every 5 steps of the input encoder the command position will change by 2 steps.

GR GEARBOX RATIO

Set gearbox ratio. In gearbox modes the ratio is specified by two arguments separated by a **/** character. The ratio is therefore specified as a fraction with the format: numerator**/**denominator. This ratio is also used for input encoder jog scaling.

Syntax	**Units**	**Range**	**to**	**Initial State**	**Privilege level**
<ad>GRnnn/nnn					
Numerator	N/A	-32768	32767 ($\pm 2^{15}$)	1	4
Denominator	N/A	1	32767 (2^{15})	1	4

Condition Requirements
Idle or Gearbox.

Notes:
Value stored in FLASH by BD command.
Not available in sequence definition – use GN & GD.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1GR2/5 Axis 1 Set electronic gearbox ratio to 2:5 - i.e. for every 5 steps of the input encoder the command position will change by 2 steps.

HC HELP WITH CONTROL MODES

A list of control modes is returned after issuing this command.

Syntax	**Units**	**Range**	**To**	**Initial State**	**Privilege level**
<ad>HC	N/A	N/A		N/A	0

Condition Requirements
None.

Notes:

Response:

```
Control Modes
 1: Servo mode
11: Open loop stepper mode
12: Checking stepper mode
13: External loop stepper mode
14: Closed loop stepper mode
```

Example:

1HC Show the help about control modes.

HD HOME TO DATUM

This command is used to find a datum point of a mechanism.
Refer to the Datum Search Strategies section of this manual and the *Datum Mode* DM command for details on datum search use.
The **HD-1** command will perform the search in the negative direction.
Soft limits are **not** used during a Home to Datum search.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>HDn	N/A	-ve	+ve	N/A	3

Condition Requirements
Idle.

Notes:

Responses:

OK	Command has been accepted.
! HARD LIMIT	Hard limit for required direction is already activated
! SOFT LIMIT	Soft limit for required direction has already been reached
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A timeout abort has been detected

Examples:

1HD	Search for datum point of axis 1 in positive direction.
1HD-1	Search for datum point of axis 1 in negative direction.

HE, HN, HP HELP

Help pages. The commands **HE** (first help page) and **HN** (help next) and **HP** (help previous) return pages showing Digiloop commands. These help pages give a concise list of the commands available and their function. It also shows the privilege level required to perform the command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>HE	N/A	N/A		N/A	0
<ad>HN	N/A	N/A		N/A	0
<ad>HP	N/A	N/A		N/A	0

Condition Requirements
None.

Notes:

Response:

```
AB                <0> Abort
AD                <4> Disable auto-execute
AE                <4> Enable auto-execute
BA                <4> Backup all
BC                <4> Backup cams
BP                <4> Backup profiles
BD                <4> Backup Digiloop parameters
BS                <4> Backup sequences
CD                <0> Clear motor datum
CP <position>        <1> Command position
CR <steps>          <1> Set creep steps
CV <speed>          <0> Constant velocity mode
DC <cam number>       <3> Define cam
DL                <2> Disable soft limits
DP <profile number>    <3> Define profile
DS <sequence number>   <3> Define sequence
EC                <0> End cam
EL                <2> Enable soft limits
EP                <0> End profile
ES                <0> End sequence
GA                <0> Absolute gearbox mode
GB                <0> Relative gearbox mode
For more help pages type: HN for next page; HP for previous page
```

Example:

1HE	Show the first help page of the controller of axis 1.

HM	HELP WITH MODE COMMANDS

This command gives an indication of the bit patterns used in the Abort Mode (**AM**) Datum Mode (**DM**) and Jog Mode (**JM**) commands.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>HM	N/A	N/A		N/A	0

Condition Requirements
None.

Notes:

Response:

```
01:
Abort Mode
AM x------- 0 = Stop Input disables control loop / 1 = stops move only
AM -x------ 0 = Stop Input latched / 1 = Stop Input momentary
AM --x----- 0 = Stall Error causes abort / 1 = doesn't cause abort
AM ---x---- 0 = Tracking Error causes abort / 1 = doesn't cause abort
AM ----x--- 0 = Timeout Error causes abort / 1 = doesn't cause abort
AM -------x 0 = Abort turns Enable Output OFF / 1 = leaves Enable Output ON
Datum Mode
DM x------- 0 = Encoder Index pulse normal / 1 = inverted
DM -x------ 0 = Datum point captured once / 1 = everytime
DM --x----- 0 = Position not changed on datum capture / 1 = set to Home
DM ---x---- 0 = Auto Direction Search disabled / 1 = enabled
DM ----x--- 0 = Auto Opposite Limit Search disabled / 1 = enabled
Jog/Joystick Mode
JM x------- 0 = Jog switch inputs disabled / 1 = enabled
JM -x------ 0 = Joystick disabled / 1 = enabled
JM --x----- 0 = Input Encoder jog disabled / 1 = enabled
JM ---x---- 0 = Select switch input disabled / 1 = enabled
```

Example:

1HM	Show the help page about mode commands.

HS	HELP WITH STATUS

This command gives an indication of the bit patterns used in the response from the Output Status (**OS**) command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>HS	N/A	N/A		N/A	0

Condition Requirements
None.

Notes:

Response:

```
Status bits
  x------- 1 = Idle
  -x------ 1 = Aborted
  --x----- 1 = Upper hard limit on
  ---x---- 1 = Lower hard limit on
  ----x--- 1 = Jog
  -----x-- 1 = On datum
01:OK
```

Example:

1HS	Show the help with status.

ID IDENTIFY

This command is used to give the type of controller and its internal software revision.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>ID	N/A	N/A		N/A	0

Condition Requirements **Notes:**
None

Response:
01:Mclennan Digiloop Motor Controller V5.16a(0.6): Multi-stepper mode (0.1)

Example:
1ID Identify controller of axis 1 (CM11).
01:Mclennan Digiloop Motor Controller V5.16a(0.6): Open loop Stepper mode

IF IF FALSE DO NEXT COMMAND

This command will examine the read port inputs and compare them with the specified bit pattern argument. If the inputs are NOT equal to the specified bit pattern (false), then the controller will execute the next command it receives, in its buffer or in its sequence. If the bit pattern IS equal (true) then the controller will skip over, i.e. not execute the next command. If the next command is skipped, the controller will give the response **'*SKIPPED*'** instead of **'*OK*'** or any other response for that command.
The bit pattern is specified as a eight digit binary number of either **0, 1** or **2** characters starting with read port 8, through to 1. A **0** defines that the input must be low (**0**), a **1** defines that the input must be high (**1**) and a **2** defines that the input is not relevant or 'don't care'. If less that eight digits are specified in the argument, then the preceding ones are assumed as low (**0**).
This command can be used to introduce a conditional response to some machine functions, and can be used to create 'clever' sequences. See also the If True (**IT**) command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>IFbbbbbbb	Bits	0	2	N/A	1

Condition Requirements **Notes:**
None

Response:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

This following sequence has been constructed to repeat a loop of moving in 400 step intervals, until read port 4 goes high (possibly activated by a switch).

1DS3	Start sequence definition
1MR400	Move 400 steps
1IF22221222	This condition is FALSE so next command is executed (i.e. NOT skipped).
1XS3	Condition was FALSE; therefore execute this sequence i.e. repeat this loop
(**1XS0)**	Return to main or another sequence (optional).
1ES	End sequence

The sequence starts by moving 400 steps. The **IF** command will then compare with the read ports, in this case it is only bit 4 that is relevant. If the condition is FALSE (the switch is not on), then the next command is executed which will restart the current sequence of move 400 steps and compare. If the condition becomes TRUE (the switch goes on), then the **XS3** command will be skipped and go on to the one after. This could be the end of the sequence or a command to then do another sequence such as returning to a master sequence.

IN INITALISE

This command will set all the programmable parameters back to their initial values, clear sequences and profiles. This is used to re-initialise all the volatile memory values to 'safe' values - e.g. if the controller was to be used in a new application. A **BA** command is required to then write these values into the *flash* non-volatile memory.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>IN	N/A	N/A		N/A	8

Condition Requirements
Idle.

Notes:

Response:

OK	Command has been accepted.

Example:

1IN	Set all parameters on axis 1 back to their initial values.

IP SET INPUT POSITION

Set the Input Encoder position value to that given in the argument.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>IPnnn	N/A	-2147483647	2147483647(±2^{32})	N/A	3

Condition Requirements
Idle, Constant velocity or Gearbox

Notes:
Value zero on power-up.

Response:

OK	Command has been accepted.

Examples:

1IP5000	Set the axis 1 Input Encoder Position to 5000.

IT IF TRUE DO NEXT COMMAND

This command will examine the read port inputs and compare them with the specified bit pattern argument. If the inputs are equal to the specified bit pattern (true), then the controller will execute the next command it receives, in its buffer or in its sequence. If the bit pattern is NOT equal (false) then the controller will skip over, i.e. not execute the next command. If the next command is skipped, the controller will give the response '***SKIPPED***' instead of '***OK***' or any other response for that command.
The bit pattern is specified as a eight digit binary number of either **0, 1** or **2** characters starting with read port 8, through to 1. A **0** defines that the input must be low (**0**), a **1** defines that the input must be high (**1**) and a **2** defines that the input is not relevant or 'don't care'. If less that eight digits are specified in the argument, then the preceding ones are assumed as low (**0**).
This command can be used to introduce a conditional response to some machine functions, and can be used to create 'clever' sequences. See also the If False (**IF**) command.

Syntax	**Units**	**Range**	**to**	**Initial State**	**Privilege level**
<ad>ITbbbbbbb	Bits	0	2	N/A	1

Condition Requirements **Notes:**
None

Response:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

If the following states are present on the inputs:

PORT :	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
STATE :	High	Low	Low	High	High	Low	Low	Low

1IT22222200	This condition is TRUE so next command is executed (i.e. NOT skipped).
1MR200	Move 200 steps
1IT22222201	This condition is FALSE so next command is skipped (i.e. is NOT executed).
1MR400	This command is skipped

If the following states are present on the inputs:

PORT :	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
STATE :	High	Low	Low	High	High	Low	Low	High

1IT22222200	This condition is FALSE so next command is skipped (i.e. is NOT executed).
1MR200	This command is skipped
1IT22222201	This condition is TRUE so next command is executed (i.e. NOT skipped).
1MR400	Move 400 steps

I.E. In the above example, read port 1 is used to select a move length and read port 2 will disable the move:

PORT :	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	
STATE :	(Ignored)	(Ignored)	(Ignored)	(Ignored)	(Ignored)	(Ignored)	Low	Low	Move 200 steps
STATE :	(Ignored)	(Ignored)	(Ignored)	(Ignored)	(Ignored)	(Ignored)	Low	High	Move 400 steps

JC SET JOYSTICK CENTRE POSITION

If the Joystick used does not have a centre tap connection then this command can be used to set the value that is the centre of the joystick and therefore zero speed. A zero value for this command indicates to the controller that the centre tap input should be used to set the joystick centre position. If the value of JC is changed while joystick control is enabled by the **JM** command then joystick operation is disabled. If joystick operation is enabled and the joystick is not at its centre position then a **! JOYSTICK FAILURE** response to the **JM** command will occur. See Joystick Calibration section.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>JCnnn	N/A	0	4095	0	4

Condition Requirements
Idle.

Notes:
Value stored in FLASH by BD command.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1JC2048	Sets centre position of the joystick input to 2048.

JM SET JOG MODE

Set the mode of operation for jog switch, joystick and encoder jog moves. There are three methods of commanding JOG or manual moves:
Jog switch inputs using fast (**SF**) and slow jog speeds (**SJ**) such as with a jog box.
Joystick input measures an analogue voltage such as from an analogue joystick or potentiometer
Input encoder jog uses the quadrature signals on the Input encoder such as with a trackerball

Syntax	Units	Range	to	Initial State	Privilege level
<ad>JMabcdefgh	Bits	0	1	10010000	6

where **a** - 0 – Jog switch inputs disabled
1 – Jog switch inputs enabled
b - 0 – Joystick input disabled
1 – Joystick input enabled
c - 0 – Input encoder jog input disabled
1 – Input encoder jog input enabled
d - 0 – Jog Select (channel increment) disabled
1 - Jog Select (channel increment) enabled
e - Reserved for future use.
f - Reserved for future use.
g - Reserved for future use.
h - Reserved for future use.

Condition Requirements
Idle

Notes:

Response:

OK	Command has been accepted.
! OUT OF RANGE	Illegal argument range or format
! JOYSTICK FAILURE	Current joystick value does not match joystick centre value within **JT**/2.

Examples:

1JM01000000	Set axis 1 to use Joystick only (jog switches and Input encoder jog disabled).

JR SET JOYSTICK RANGE

This command is used to set the range of the joystick input between the threshold (set by the **JT** command) and the maximum speed (set by the **JS** command). See Joystick Calibration section.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>JRnnn	N/A	100	4095	320	4

Condition Requirements
Idle.

Notes:
Value stored in FLASH by **BD** command.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Value is above valid range.
! JOYSTICK FAILURE	Value is below 100.

Example:

1JR500	Sets usable range of the joystick input to 500.

JS SET JOYSTICK SPEED

Set the normal speed for all following manual joystick moves. Please note that few joysticks can achieve full voltage swing and therefore this sets the speed for full movement of the joystick. See Joystick Calibration section.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>JSnnn	Steps/sec	1	400000	10000	4

Condition Requirements
Idle.

Notes:
Value stored in FLASH by **BD** command.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1JS2000	Sets jog speed of axis 1 controller to 2000 Steps/sec.

JT SET JOYSTICK THRESHOLD

This command can be used to set the joystick value below which the joystick is inactive. If there is a zero threshold value the controller would be in joystick jogging mode continuously. See Joystick Calibration section.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>JTnnn	N/A	1	4095	40	4

Condition Requirements
Idle.

Notes:
Value stored in FLASH by **BD** command.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1JT50	Sets joystick input threshold to 50.

KF SET FEEDFORWARD COEFFICIENT

Set velocity feedforward servo coefficient. This compensates for the position offset caused by the velocity lag introduced by **KV**. For positioning moves **KF** is normally set at zero, but for Profiles and Cam moves where the actual position should not lag behind the command position, **KF** should be set equal to **KV**.
In dual encoder feedback systems **KX** also causes a velocity lag. The value of complete **KF** compensation needed is equal to **KX** multiplied by the ratio of Auxiliary encoder pulses to Position Encoder pulses plus the value of **KV**. It is not usually necessary for complete compensation of the velocity lag as this adversely effects the settling time of the system.
This command is only appropriate in the servo motor control mode (not for stepper motor control)

Syntax	Units	Range	to	Initial State	Privilege level
<ad>KFnnn	Number	0	32767	0	7

Condition Requirements
Servo mode, Idle

Notes:
Value stored in FLASH by **BD** command.

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Examples:

1KF500	Set velocity feedforward on axis 1 to 500.

KP (IN SERVO MODE) SET PROPORTIONAL GAIN COEFFICIENT

Set proportional gain servo coefficient. The stiffness and accuracy of the servo loop are controlled by the magnitude of the proportional gain.
See next command description for use in stepper motor mode

Syntax	Units	Range	to	Initial State	Privilege level
<ad>KPnnn	Number	0	32767	10	7

Condition Requirements	**Notes:**
Idle	Value stored in FLASH by BD command.

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Examples:

1KP100	Set the proportional gain on axis 1 to 100.

KP (IN STEPPER MODE) SET PROPORTIONAL GAIN COEFFICIENT (CORRECTION GAIN)

Set proportional gain for auto-correction moves when in closed-loop stepper mode (**CM14**). The correction gain is expressed as a percentage. The amount of attempted correction for each iteration is the difference between the Command Position and the encoder read Actual Position, scaled by this value. If the result is less than one step then it will use one step of correction. Each successful iteration, separated by settling time, should therefore result in less of an error and the next attempt will be less. Error correction will continue until the error is within the window (**WI**) or the Time Out (**TO**) period has expired.
Care must be taken with remote feedback encoders, not to set too higher value that might give rise to an oscillatory system.
See previous command description for use in servo mode

Syntax	Units	Range	to	Initial State	Privilege level
<ad>KPnnn	%	0	100	70	7

Condition Requirements	**Notes:**
Idle	Value stored in FLASH by BD command.

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Examples:

1KP100	Set the proportional gain on axis 1 to 100.

KS SET SUM GAIN COEFFICIENT

The Sum servo coefficient is the sum of the integral and proportional components of the servo control loop. The accuracy of the servo loop depends on having a non-zero value of **KS** at the expense of transient response.
This command is only appropriate in the servo motor control mode (not for stepper motor control)

Syntax	Units	Range	to	Initial State	Privilege level
<ad>KSnnn	Number	0	32767	0	7

Condition Requirements
Servo mode, Idle

Notes:
Value stored in FLASH by BD command.

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Examples:

1KS50	Set the Sum gain on axis 1 to 50.

KV SET VELOCITY FEEDBACK COEFFICIENT

The value of this coefficient defines the magnitude of the velocity feedback signal derived from the position encoder. This coefficient influences the transient response by producing a damping effect. It effects the system by reducing overshoot and enhancing stability, but too high a value can create high frequency instability.
This command is only used in the servo motor control mode (not for stepper motor control)

Syntax	Units	Range	to	Initial State	Privilege level
<ad>KVnnn	Number	0	32767	0	7

Condition Requirements
Servo mode, Idle

Notes:
Value stored in FLASH by BD command.

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Examples:

1KV500	Set the Velocity feedback on axis 1 to 500.

KX SET EXTRA VELOCITY FEEDBACK COEFFICIENT

The Extra Velocity Feedback coefficient. It is used in Dual Encoder feedback mode. The value of this coefficient defines the magnitude of the velocity feedback signal derived from the auxiliary (third) encoder. This coefficient influences the system transient response by producing a damping effect.
This command is only appropriate in the servo motor control mode (not for stepper motor control)

Syntax	Units	Range	to	Initial State	Privilege level
<ad>KXnnn	Number	0	32767	0	7

Condition Requirements	**Notes:**
Servo mode, Idle	Value stored in FLASH by BD command.

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Examples:

1KX5000	Set the Extra Velocity feedback on axis 1 to 5000.

LC LIST CAM

This command will list a previously defined Cam profile.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>LCn	Cam Number	0	7	N/A	0

Condition Requirements	**Notes:**
Servo mode, Idle	

Responses

The command will either respond with the axis address identifier followed by each line of the Cam definition Profile, or an error message:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.
! CAM UNDEFINED	Cam has not been defined yet.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

A controller that had previously been programmed with:

1DC3	Start Cam 3 definition.
1XY200/500	First Cam Profile point.
1XY400/500	Next Cam Profile point.
1XY600/-200	Next Cam Profile point.
1XY700/-200	Next Cam Profile point.
1XY750/0	Next Cam Profile point.
1EC	End of Cam profile definition.

The command 1LC3 would give:

```
01:Cam 3:
XY    0 /    0
XY  200 /  500
XY  400 /  500
XY  600 / -200
XY  700 / -200
XY  750 /    0
```

LD SET LIMIT DECELERATION

Set the deceleration rate for stopping when hitting a Hard Limit. It is also used during a stop such as after a Control C. This value would normally be set to a high value to prevent limit overrun, but can be used to reduce the harshness of stopping on a limit. This should also be used to prevent stepper motors desynchronising and losing steps through excessive deceleration.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>LDnnn	Steps/sec^2	1	20000000	2000000 (servo)	4
				50000 (stepper)	

Condition Requirements
Idle

Notes:
Value stored in FLASH by BD command.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1LD1000000 Sets Limit Deceleration of axis 1 controller to 1000000 Steps/sec^2.

LL SET LOWER SOFT LIMIT POSITION

This command will set the Lower Soft Limit Position to the value given in the argument. Subsequent moves by the Move Absolute (**MA**) or Move Relative (**MR**), constant velocity (**CV**) and manual Jog moves will not be allowed below this Lower Limit if the Soft Limits are enabled.

Syntax	Units	Range	To	Initial State	Privilege level
<ad>LLnnn	Steps	-2147483647	Upper Soft Limit	-2000000000	3

Condition Requirements
Idle

Notes:
Value stored in FLASH by BD command.

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.
! LIMITS CONFLICT	Attempting to set lower limit above or equal to upper limit

Example:

1LL-4000 Set the axis 1 Lower Soft Limit Position to -4000.

LP LIST PROFILE

This command will list a previously defined velocity Profile.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>LPn	Profile No.	0	7	N/A	0

Condition Requirements
Servo mode, Idle

Notes:

Responses

The command will either respond with the axis address identifier followed by each line of the Profile definition, or an error message:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.
! PROFILE UNDEFINED	Profile has not been defined yet.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

A controller that had previously been programmed with:

1DP6	Start definition of Profile.
1MR2000	First move.
1MR7000	Next move.
1MR1000	Next move.
1MR0	Next move.
1EP	End of Profile definition.

The command 1LP6 would give:

```
01:Profile 6:
MR      2000
MR      7000
MR      1000
MR         0
```

LS LIST SEQUENCE

This command will list a previously defined Sequence.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>LSn	Seq. Number	0	7	N/A	0

Condition Requirements **Notes:**

Idle

Responses

The command will either respond with the axis address identifier followed by each line of the Sequence definition, or an error message:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.
! SEQUENCE UNDEFINED	Sequence has not been defined yet.

Example:

A controller that had previously been programmed with:

1DS2	Start definition of sequence 2.
1MA2000	First move (absolute).
1MR7000	Next move (relative).
1DE1000	Delay for 1 second.
1MA0	Next move (return to start position).
1XS2	Execute sequence 2 (loop to start of this sequence).
1ES	End of sequence definition.

The command 1LS2 would give:

```
01:Sequence 2:
 MA    2000
 MR    7000
 DE    1000
 MA       0
 XS       2
```

MA MOVE TO ABSOLUTE POSITION

This command will move the motor to the position given in the argument. This position is relative to the Command Position of zero.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>MAnnn	Steps	-2147483647	2147483647 ($\pm 2^{32}$)	N/A	1

Condition Requirements **Notes:**

Idle

Responses

OK	Command has been accepted.
! HARD LIMIT	Hard limit for required direction is already activated
! SOFT LIMIT	Move attempted that exceeds the Soft limit in the required direction
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A timeout abort has been detected

Example:

If axis 1 has a current Command Position of 5000 then the command:

1MA4000 Will move 1000 steps in the negative direction to arrive at a Command position of 4000.

MD MOVE TO DATUM POSITION

This command will move the motor to the datum position if one has already been captured.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>MD	N/A	N/A		N/A	1

Condition Requirements **Notes:**
Idle, Valid captured datum.

Responses

OK	Command has been accepted.
! NO VALID DATUM	A datum point has yet to be found
! HARD LIMIT	Hard limit for required direction is already activated
! SOFT LIMIT	Move attempted that exceeds the Soft limit in the required direction
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A timeout abort has been detected

Example:
If axis 1 has a current valid Datum Position of 12496 then the command:
1MD Will move to the position of 12496.

MR MOVE TO RELATIVE POSITION

This command will move the motor to the position given in the argument relative to the current Command Position.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>MRnnn	Steps	-2147483647	2147483647 (±2^{32})	N/A	1

Condition Requirements **Notes:**
Idle

Responses

OK	Command has been accepted.
! HARD LIMIT	Hard limit for required direction is already activated
! SOFT LIMIT	Move attempted that exceeds the Soft limit in the required direction
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A timeout abort has been detected

Example:
If axis 1 has a current Command Position of 5000 then the command:
1MR4000 Will move 4000 steps in the positive direction to arrive at a Command position of 9000.

NC NUMBER OF CHANNELS

This command will set the maximum number of channels of a multi-channel system to the argument given. This is used to avoid selecting a channel beyond the number of channels connected to the system.
It also sets the maximum channel number, when using the channel increment input, before returning to zero.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>NCn	Channel No.	Currently Selected Channel	8	8	8

Condition Requirements **Notes:**
Idle

Responses:

`OK`	Command has been accepted.
`! OUT OF RANGE`	Argument is out of valid range.
`! CHANNELS CONFLICT`	Number of channels less than selected channel

Example:

1NC4	Set maximum number of channels for axis 1 to 4.

NP SET NEW PIN SECURITY NUMBER

Set the PIN security number to that given in the argument. You must enter the existing PIN using the PI command first.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>NPnnn	Number	0000	9999	9999	9

Condition Requirements **Notes:**
Idle

Response:
OK Command has been accepted.

Examples:

1PI0001	Enter existing PIN number.
1NP0666	Set the axis 1 PIN number to 0666.

OA OUTPUT ACTUAL POSITION

This command will give the current Actual Position read from the position encoder (encoder 1). This position is derived from the incoming position encoder pulses (scaled by the encoder ratio (**ER**).

Syntax	Units	Range	to	Initial State	Privilege level
<ad>OA	N/A	N/A		N/A	0

Condition Requirements **Notes:**
None

Response:
The response is a string of numeric characters. If the communications are in Verbose Mode, the reply is preceded by **Actual pos =**

Example:
If the controller of axis 1 currently has an Actual Position of 70551 then the command:

1OA	in Verbose Mode will respond:	**01:Actual pos = 70551**
1OA	in Quiet Mode will respond:	**01:70551**

OC OUTPUT COMMAND POSITION

This command will give the current Command Position.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>OC	N/A	N/A		N/A	0

Condition Requirements **Notes:**
None.

Response:
The response is a string of numeric characters. If the communications are in Verbose Mode, the reply is preceded by **Command pos =**

Example:
If the controller of axis 1 currently has a Command Position of 45280 then the command:

1OC	in Verbose Mode will respond:	**01:Command pos = 45280**
1OC	in Quiet Mode will respond:	**01:45280**

OD OUTPUT CAPTURED DATUM POSITION

This command will give the current captured datum position.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>OD	N/A	N/A		N/A	0

Condition Requirements **Notes:**
None.

Response:
The response is a string of numeric characters. If the communications are in Verbose Mode, the reply is preceded by **Datum position =**
! NO VALID DATUM if a datum has not been captured.

Example:
If the controller of axis 1 currently has a datum position of 28456 then the command:

1OD	in Verbose Mode will respond:	**01:Datum position = 28456**
1OD	in Quiet Mode will respond:	**01:28456**

OF OUTPUT FOLLOWING ERROR BETWEEN COMMAND AND ACTUAL POSITIONS

This command will give the difference between the current Command Position and the current encoder read Actual Position. Numerically it is the Command Position (**CP**) - Actual Position (**AP**).

Syntax	Units	Range	To	Initial State	Privilege level
<ad>OF	N/A	N/A		N/A	0

Condition Requirements **Notes:**
None.

Response:
The response is a string of numeric characters. If the communications are in Verbose Mode, the reply is preceded by **Following error =**

Example:
If the controller of axis 1 currently has a Current position of 1000 and an Actual Position of 1050 then the command:

1OF	in Verbose Mode will respond:	**01:Following error = -50**
1OF	in Quiet Mode will respond:	**01:-50**

OI OUTPUT INPUT POSITION

This command will give the current value of the Input Position, read from encoder 3.

Syntax	Units	Range	To	Initial State	Privilege level
<ad>OI	N/A	N/A		N/A	0

Condition Requirements
None

Notes:
Used in *electronic gearbox* and *Cam profiles* etc.

Responses:
The response is a string of numeric characters. If the communications are in Verbose Mode, the reply is preceded by **Input pos =**

Example:
If the controller of axis 1 currently has an Input Position of 30401 then the command:
1OI in Verbose Mode will respond: **01:Input pos = 30401**
1OI in Quiet Mode will respond: **01:30401**

OS OUTPUT STATUS

This command will give a binary string that will represent the current status of the controller.

Syntax	Units	Range	To	Initial State	Privilege level
<ad>OS	N/A	N/A		N/A	0

Condition Requirements
None

Notes:

Response:
The response is a string of 8 numeric characters of either 0 or 1. If the communications are in Verbose Mode, the reply is preceded by **Status = .** Each bit is described as follows:

Status = abcdefgh	where:	
a -	0 –	Controller is busy (doing something)
	1 –	Controller is idle
b -	0 –	OK
	1 –	Error (abort, tracking, stall, timeout etc.)
c -	0 –	Upper hard limit is OK
	1 –	Upper hard limit is ON
d -	0 –	Lower hard limit is OK
	1 –	Lower hard limit is ON
e -	0 –	Not jogging or joystick moving
	1 –	Jogging or joystick moving
f -	0 –	Not at datum sensor point
	1 –	On datum sensor point
g -	0 –	For future use
	1 –	For future use
h -	0 –	For future use
	1 –	For future use

Example:
If the controller of axis 1 is currently moving to a position (using a **MA** command):
1OS in Verbose Mode will respond: **01:Status = 00000000**
1OS in Quiet Mode will respond: **01:00000000**
If the controller of axis 1 is currently stopped on the upper hard limit:
1OS in Verbose Mode will respond: **01:Status = 10100000**
1OS in Quiet Mode will respond: **01:10100000**

OT OUTPUT (THIRD) AUXILIARY POSITION

This command will give the current encoder read Auxiliary Position (encoder 2). This position is derived from the incoming position encoder pulses of the second encoder (dual encoder feedback).

Syntax	Units	Range	to	Initial State	Privilege level
<ad>OT	N/A	N/A		N/A	0

Condition Requirements **Notes:**
None

Response:
The response is a string of numeric characters. If the communications are in Verbose Mode, the reply is preceded by **Auxiliary pos =**

Example:
If the controller of axis 1 currently has an Auxiliary Position of 20501 then the command:

1OT	in Verbose Mode will respond:	**01:Auxiliary pos = 20501**
1OT	in Quiet Mode will respond:	**01:20501**

OV OUTPUT VELOCITY

This command will give the current velocity of the Actual Position (position encoder), unless in open loop stepper mode where the velocity of the command position is used. This value is averaged over the time given in the argument in milliseconds. You would therefore choose a time to give the accuracy you require at the expense of the time to complete the command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>OVnnn	ms	1	10000	N/A	0

Condition Requirements **Notes:**
None

Response:
The response is a string of numeric characters. If the communications are in Verbose Mode, the reply is preceded by **Velocity =**

Example:
If the controller of axis 1 currently at 20000 steps per second but instantaneously currently very slightly lagging then:

1OV250	in Verbose Mode will respond:	**01:Velocity = 19994**
1OV250	in Quiet Mode will respond:	**01:19994**

Notice that as the average time is a quarter of a second, then the speed is a multiple of four.

PI ENTER PIN NUMBER

This command allows you to enter the PIN security number. This gives full access to all commands and allows the Privilege Level to be changed using the (**PL**) command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>Pinnnn	N/A	0000	9999	N/A	0

Condition Requirements
None

Notes:
The full access may be cancelled either by entering an invalid PIN or switching off.

Response:

OK	Command has been accepted.
! INVALID PIN	Wrong PIN entered.

Example:
If the controller of axis 1 currently has a security PIN number of 4423
1PI4423 will allow the privilege level to be changed.

PL SET PRIVILEGE LEVEL

Set the privilege level. This command allows you to set a privilege level that will restrict the commands available to the user. This can be used to prevent accidental changing of important set-up parameters.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>PLn	N/A	0	9	8	0

Condition Requirements
The PIN number must have been entered (PI)

Notes:
Value stored in FLASH by BD command.

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.
! PRIVILEGE VIOLATION	PIN not entered to allow changes in privilege level.

Example:
1PL1 Sets the privilege level to 1 (queries and moves only).

PT PROFILE TIME

This command allows you to enter the time to complete each element in a profile definition.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>PTnnn	ms	0	32000	1000	4

Condition Requirements
Servo mode, Idle

Notes:

Response:

OK	Command has been accepted.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:
1PT50 will set the time for each element of the profile to be 50 ms

QA QUERY ALL PARAMETERS

Query All. Returns all of the current settings and modes of the controller along with the current positions in a single page format.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>QA	N/A	N/A		N/A	0

Condition Requirements None

Notes:

Response:
The response is alpha-numeric strings of characters. Each line gives the parameter names and their values. See example for the format.

Example:
1QA Will generate a response of the form:

```
Mclennan Digiloop Motor Controller V5.16a(0.6): Multi-stepper mode (0.1)
Address = 1                    Privilege level = 8
Mode = Idle                    Channel = 1 of 8
Kf = 0      Kp = 70      Ks = 0      Kv = 0      Kx = 0
Slew speed = 1000              Limit decel = 50000
Acceleration = 2000            Deceleration = 3000
Creep speed = 800              Creep steps = 10
Jog speed = 100                Fast jog speed = 500
Joystick speed = 10000
Settling time = 100            Backoff steps = 0
Window = 4                     Threshold = 50 %
Tracking = 4000                Timeout = 8000
Lower soft limit = -2000000000    Upper soft limit = 2000000000
Lower hard limit off           Upper hard limit off
Jog enabled                    Joystick disabled
Gearbox ratio =    1/1         Encoder ratio = -1/1
Command pos = 47592                Actual pos = 47592
Input pos = 558                Home pos = 0
Pos error = 0                  Datum pos = None
Valid sequences: 0 1 (Autoexec 0 enabled)
Valid cams: 0 1
Valid profiles: 0 (Profile time = 50 ms)
Read port: 10101010            Last write: 00000000
```

QC QUERY CURRENTLY SELECTED CHANNEL NUMBER

This command will give the currently selected channel number set by the **CH** command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>QC	N/A	N/A		N/A	0

Condition Requirements PM608 only

Notes:

Responses:
The response is a numeric character following the axis address identifier.

Example:
If the controller of axis 1 currently has channel 3 selected, then the command:
1QC will respond:
01:3 in quiet mode
or **01:Channel = 1** in verbose mode

QD QUERY DATA

This command will give the data on the current position and status. The data is returned as a HEX string that is decoded by external computer programs. The format of the data is subject to change so it is not defined in this manual.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>QD	N/A	N/A		N/A	0

Condition Requirements
None

Notes:

Responses:
The response is an alpha-numeric string of characters showing all the current position and status of the controller.

QJ QUERY JOYSTICK SETTINGS

Query the current settings of the Joystick parameters. Returns the current values for Joystick Speed (JS), Joystick Centre (JC), Joystick Threshold (JT) and Joystick Range (JR)

Syntax	Units	Range	to	Initial State	Privilege level
<ad>QJ	N/A	N/A		N/A	0

Condition Requirements
None

Notes:

Response:
. See example for the format.

Example:

1QJ	Will generate a response of the form: **01:JS = 10000 JC = 0 JT = 40 JR = 320**

QK QUERY K COEFFICIENTS

Query servo loop coefficients. Returns the current settings of the KF, KP, KS, KV and KX coefficients.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>QK	N/A	N/A		N/A	0

Condition Requirements
None

Notes:

Response:
The response is an alpha-numeric string of characters showing the parameter name and its value. See example for the format.

Example:

1IN	Set to initial values.
1KP2909	Set proportional gain to 2909.
1KV357	Set velocity feedback to 357.
1KS3258	Set Sum coefficient to 3258.
1QK	Will generate a response of the form: **01:Kf = 0 Kp = 2909 Ks = 3258 Kv = 0 Kx = 0**

QL QUERY CURRENT PRIVILEGE LEVEL

This command will give the current privilege level. The higher the level, the more commands you can use.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>QL	N/A	N/A		N/A	0

Condition Requirements **Notes:**
None

Responses:
The response is a numeric character.

Example:
If the controller of axis 1 currently has a privilege level of, then the command:
1QL will respond: **01:Privilege level = 6**

QM QUERY MODES

This command will give the current Control Mode (**CM**), Abort Mode (**AM**), Datum Mode (**DM**), and Jog Mode (**JM**).

Syntax	Units	Range	to	Initial State	Privilege level
<ad>QM	N/A	N/A		N/A	0

Condition Requirements **Notes:**
None

Responses:
The response is the axis address identifier, followed by the following (see example for format):
CM = currently set control mode (in decimal)
AM = currently set abort mode (as binary bit pattern)
DM = currently set datum mode (as binary bit pattern)
JM = currently set jog mode (as binary bit pattern)

Example:
If the controller of axis 1 is set to servo motor controller
1QM may give a response of:
01:CM = 11 AM = 00000000 DM = 00000000 JM = 10010000

QN QUERY NUMBER OF CHANNELS

This command will give the current number of channels set by the **NC** command.

Syntax	Units	Range	To	Initial State	Privilege level
<ad>QN	N/A	N/A		N/A	0

Condition Requirements **Notes:**
PM608 only

Responses:
The response is a numeric character following the axis address identifier.

Example:
If the controller of axis 1 currently has the number of channels set to 7, then the command:
1QN will respond:
01:7 in quiet mode
or **01:Number of channels = 7** in verbose mode

QP QUERY POSITIONS

Query the current position information. Returns the current values for Command Position (CP), Actual Position (AP), Input (IP) Position Auxiliary Position (TP) and Datum Position (OD)

Syntax	Units	Range	to	Initial State	Privilege level
<ad>QP	N/A	N/A		N/A	0

Condition Requirements **Notes:**
None

Response:
The response is an alpha-numeric string of characters showing all the current position variables. See example for the format.

Example:

1QP	Will generate a response of the form:

01:CP = -1026 AP = -1026 IP = 1050 TP = 0 OD = -2050

QS QUERY SPEEDS

Query the current settings for the speeds and accelerations. Returns the current settings of SV, SC, SA, SD and LD.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>QS	N/A	N/A		N/A	0

Condition Requirements **Notes:**
None

Response:
The response is an alpha-numeric string of characters showing all the speed related variables. See example for the format.

Example:

1SC700	Set creep speed to 700 steps/sec.
1SV16200	Set slew speed to 16200 steps/sec.
1SA50000	Set deceleration to 50,000 steps/sec^2.
1SD100000	Set deceleration to 100,000 steps/sec^2.
1SD200000	Set limit deceleration to 200,000 steps/sec^2.
1QS	Will generate a response of the form:
	01:SC = 700 SV = 16200 SA = 50000 SD = 100000 LD = 200000

RP READ INPUT PORT

This command will examine the read port inputs and return their current state as an eight digit numeric string of either **0** or **1** characters. The string starts with read port 8. A **0** indicates that the input is low (0V or open-circuit) and a **1** indicates that the input is high (+24V).

Syntax	Units	Range	to	Initial State	Privilege level
<ad>RP	N/A	N/A		N/A	0

Condition Requirements
None

Notes:
If an **RP** command is executed with the read ports open circuit, a reply of **00000000** will be returned

Responses
The response is a string of 8 numeric characters of either 0 or 1. If the communications are in Verbose Mode, the reply is preceded by Port: **.**

Example:
If the following states are present on the inputs:

PORT :	8	7	6	5	4	3	2	1
STATE :	LOW	LOW	LOW	HIGH	LOW	LOW	LOW	HIGH

Then

1RP	in Verbose Mode will respond:	**01:Port: 00010001**
1RP	in Quiet Mode will respond:	**01: 00010001**

RS RESET

This command will reset the *tracking abort, stall abort, time out abort* or *user (command) abort* conditions. The enable output and the servo control loop will be reset. It will also set the Command position to be equal to the Actual position (except open-loop stepper mode).

Syntax	Units	Range	to	Initial State	Privilege level
<ad>RE	N/A	N/A		N/A	3

Condition Requirements
None

Notes:

Responses

RESET or ! NOT ABORTED	Command has been accepted
! INPUT ABORT	Stop Input still active.

Example:

1RS	Reset abort on axis 1 controller.

SA SET ACCELERATION

Set the acceleration rate for changes of velocity for all following moves.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>SAnnn	Steps/sec^2	1	20000000	2000	4

Condition Requirements
Idle

Notes:
Value stored in FLASH by BD command.

Condition Requirements

Notes:

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1SA10000 Sets acceleration of axis 1 controller to 10000 Steps/sec^2.

SC SET CREEP SPEED

Set the *creep speed* for all following moves. This is the speed that at which moves with a non-zero creep distance will stop. It is the speed that slow datum search will be moved at (**HD** command). In stepper modes the *creep speed* is equivalent to the *base* speed, i.e. the speed that moves will start and stop at.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>SCnnn	Steps/sec	1	400000 in servo mode	800	4
			800 in stepper modes		

Condition Requirements
Idle

Notes:
Value stored in FLASH by BD command.
MUST NOT BE HIGHER THAN SLEW SPEED (SV)

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range or greater than current slew speed (SV).

Example:

1SC700 Sets creep speed of axis 1 controller to 700 Steps/sec.

SD SET DECELERATION

Set the deceleration rate for changes of velocity for all following moves.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>SDnnn	Steps/sec^2	1	20000000	3000	4

Condition Requirements
Idle

Notes:
Value stored in FLASH by BD command.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1SD100000 Sets deceleration of axis 1 controller to 100000 Steps/sec^2.

SE SET SETTLING TIME

Set the settling time for all following moves. The settling time operation depends on the PM600's control mode.

CM1 Servo mode
The settling time elapses at the end of each move to allow the motor to settle. The end of a move is defined by the **OF** (following error or position difference) value being less than the **WI** (end of move window) value for the **SE** (settling) time. If the following error exceeds the window, then the settling counter will be reset and therefore it must be within the window for the whole length of the settling time.

CM11 Open loop stepper mode
The settling time elapses between the end of each move and the next. The end of the previous move is simply the end of the move profile defined by the command position profile. This allows for mechanical oscillations to cease.

CM12 Checking stepper mode
The settling time elapses at the end of each move to allow the motor to settle before the 'move complete' test occurs. The move is defined as being 'complete' if the **OF** (following error or position difference) value is less than the **WI** (end of move window) value.

CM13 External loop stepper mode
The operation is the same as in described in servo mode.

CM14 Closed loop stepper mode
The settling time elapses at the end of each move to allow the motor to settle before the 'move complete' test occurs. The move is defined as being 'complete' if the **OF** (following error or position difference) value is less than the **WI** (end of move window) value. If the position error is greater than the end of move window then a correction move will take place.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>SEnnn	milliseconds	0	20000	100	6

Condition Requirements
Idle

Notes:
Value stored in FLASH by BD command.

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1SE1000	Sets settling time of axis 1 controller to 1 second.

SF SET FAST JOG SPEED

Set the fast speed for all following manual *jog switch* moves. The jog movement will accelerate up to this speed when a jog input and the jog fast inputs are active.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>SFnnn	Steps/sec	1	200000	500	4

Condition Requirements
Idle.

Notes:
Value stored in FLASH by BD command.

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1JF1000	Sets fast jog speed of axis 1 controller to 1000 Steps/sec.

SH SET HOME POSITION

Set the Home position value to that given in the argument. The Home Position can be used during a datum search to automatically set the datum point to the given value, when using the Home to Datum (**HD**) command, if the correct Datum Mode is set (see Datum Search section and **DM** command).

Syntax	Units	Range	to	Initial State	Privilege level
<ad>SHnnn	Steps	-2147483647	2147483647 (±2^{32})	0	6

Condition Requirements
Idle.

Notes:
Value stored in FLASH by BD command.

Response:

OK	Command has been accepted.

Examples:

1SH-34277	Set the axis 1 Home Position to -34277.

SJ SET JOG SPEED

Set the normal speed for all following manual *jog switch* moves. The jog movement will be at this speed when a jog input is active, but not the jog fast input.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>SJnnn	Steps/sec	1	20000	100	4

Condition Requirements
None.

Notes:
Value stored in FLASH by BD command.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1SJ50	Sets jog speed of axis 1 controller to 50 Steps/sec.

SL ENABLE/DISABLE SOFT LIMITS

This command is used to enable or disable the soft limit protection. If the soft limits are disabled, further movement is NOT bounded by the upper and lower soft limits. Hard limits will still be active and cannot be disabled.

Syntax	Units	Range	to	Initial Value	Privilege level
<ad>SLb	N/A	0 –Soft limits disabled	1 – Soft limits enabled	Enabled (1)	6

Condition Requirements
Idle.

Notes:
Value stored in FLASH by BD command.

Response:

OK	Command has been accepted.

Example:

1SL0	Sets the soft limits OFF (disabled) for controller axis 1.
1SL1	Sets the soft limits ON (enabled) for controller axis 1.

ST STOP

This command will stop any current move, decelerate the motor speed down at the **SD** rate, then stop and return to *idle* mode.
This command is buffered and is only responded to when it reached in the command queue. Care must therefore be taken that there are no commands that hold up the queue between the move command and the **ST** command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>ST	N/A	N/A		N/A	0

Condition Requirements
Not Idle

Notes:
Will exit constant velocity mode or gearbox mode.

Responses

OK	Command has been accepted.
! NOT ALLOWED IN THIS MODE	The controller is already stopped (idle).

Example:

1CV1000	Will start axis 1 moving in constant velocity mode (1000 steps/sec).
1ST	This will then stop the current move of axis 1.

SV SET VELOCITY

Set the Slew (maximum) velocity for all following moves.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>SVnnn	Steps/sec	1	1,200,000	1000	4

Condition Requirements
Idle

Notes:
Value stored in FLASH by BD command.
MUST BE NO LESS THAN CREEP SPEED (SC)

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range or less than current creep speed (SC).

Example:

1SV5000	Sets slew speed of axis 1 controller to 5000 Steps/sec.

TH SET THRESHOLD

This command will set the motor stalled threshold. Failure of an encoder is indistinguishable from a stalled motor, and messages from the PM600 refer to *stall abort* rather than encoder failure.
A stalled motor (or encoder failure) is detected by looking for changes in the position encoder signals (or equivalently the changes in observed motor position). If the motor does not move, and the voltage output value from the PM600 exceeds the value set by the **TH** command for a time of 256ms, then the PM600 will set its output to zero and set Stall Abort condition. The threshold is expressed as a percentage of full scale output of the Analogue output.
The servo system will have coulomb friction and the voltage required to overcome this friction, varies from system to system, so the value of **TH** must be large enough not to nuisance trigger but small enough to detect any failure.

If a *stall abort* condition occurs, the front panel status display shows a S, and movement is stopped. Subsequent moves will not function but will return the response **! STALL ABORT** until reset by either a Reset (**RS**) command or by powering off. The stall abort function can be enabled or inhibited by using the **AM** (abort mode) command.
The response to a **CO** command is **! STALL ABORT**.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>THnnn	%	1	100	50	6

Condition Requirements
Idle.

Notes:
Value stored in FLASH by BD command.
Not used for stepper modes.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1TH40	Set the Threshold before *motor stalled* condition for axis 1 to 40%.

TO SET TIME-OUT/NOT COMPLETE TIME

This command will set the Not Complete/Time-Out time. This is the maximum time allowed at the end of a move, from when the Command Position reaches its target, until the move has settled and completed. If either the move or error correction is not completed within this time then a Time Out will be detected and the controller will Abort if set if set to do so using the Abort Mode (**AM**) command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>TOnnn	milliseconds	1	60000	8000	6

Condition Requirements
Idle.

Notes:
Value stored in FLASH by BD command.
Not used for open-loop stepper modes.

Response:

OK	Command has been accepted.

Examples:

1TO4000	Set the axis 1 Time out to 4 seconds (4000mS).

TP SET (THIRD) AUXILIARY POSITION

Set the Auxiliary (third) Position value to that given in the argument. This position is derived from the incoming position encoder pulses of the second encoder (dual encoder feedback).

Syntax	Units	Range	to	Initial State	Privilege level
<ad>TPnnn	Steps	-2147483647	2147483647 ($\pm 2^{32}$)	N/A	3

Condition Requirements
Idle, Constant velocity or gearbox

Notes:
Value zero on power-up.

Response:

OK	Command has been accepted.

Examples:

1TP-5000	Set the axis 1 Auxiliary Position to -5000.

TR SET TRACKING WINDOW

This command will set the tracking window. The *Tracking window* is the allowable difference between the *Command Position* and the *Actual Position*. When the motor is stationary this is the allowable static error. During a move, a changing *command position* is generated. The *Tracking Window* operates on the difference between the *actual position* and this moving *command position*. The servo system will have a *following error,* so the value of **TR** must be large enough not to nuisance trigger but small enough to detect any failure.

If the *tracking* window is exceeded the front panel display will show a and if abort is enabled the Error output signal will be activated and the controller will *abort* any moves.
The abort function can be enabled or inhibited by using the **AM** (abort mode) command. If aborted, subsequent moves will not function but will return the response **! TRACKING ABORT** until reset by either a Reset (**RS**) command or by powering off.

Syntax	Units	Range	To	Initial State	Privilege level
<ad>TRnnn	Steps	0	2147483647 (2^{32})	4000	6

Condition Requirements
Idle

Notes:
Value stored in FLASH by BD command
Not used for open-loop stepper modes.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1TR400	Set the Tracking Window for axis 1 to 400 steps.

TUNE TUNE SERVO COEFFICIENTS

Invoking the TUNE command can usually derive an approximate set of servo coefficients. The controller will *exercise* the motor over a small displacement for a few seconds and obtain a set of values for the *K* coefficients that should be stable and provide a reasonable disturbance rejection.
The tuning algorithm can fail if there is excessive backlash, if the low frequency loop gain is either very small or very large or the feedback encoder phasing is wrong. Further optimisation of system response may be required to achieve the desired performance.
The **TUNE** command only affects **KP, KV, KS and KV** therefore its use in a double encoder system is inappropriate and will produce a **! TUNE FAILURE** error.
The TUNE command is only appropriate in the servo motor control mode (not for stepper motor control)

Syntax	Units	Range	To	Initial State	Privilege level
<ad>TUNE	N/A	N/A		N/A	7

Condition Requirements
Servo mode, Idle

Notes:
Not used for stepper modes.

Responses:

In *verbose* mode, if the *tune* is successful, the first response will be
01: Reversals = nn Amplitude = mm
nn and mm are parameters relating to the system response. These parameters are used by optimisation programs. Shortly afterwards the values of the coefficients will be shown.
01:Kf = 0 Kp = 2909 Ks = 3258 Kv = 0 Kx = 0
In *quiet* mode, if the *tune* is successful, the response will be
01:OK
If the *tune* fails the response in both modes will be
01:! TUNE FAILURE

Other Responses

! HARD LIMIT	Hard limit for required direction is already activated
! SOFT LIMIT	Soft limit for required direction has already been reached
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A timeout abort has been detected
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

1TUNE	Tune coefficients on axis 1 controller.

UC UNDEFINE CAM

This command will undefine or cancel a Cam definition. **Note** that this will only remove the cam definition from the volatile memory and to change the non-volatile flash memory this command must be followed by a backup cam (**BC**) command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>UC	Cam number	0	7	N/A	5

Condition Requirements	Notes:
Servo mode, Idle.	Value stored in FLASH by BC command. Not used for stepper modes.

Responses

OK	Command has been accepted.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

1UC	Delete Cam from axis 1 controller.

UL SET UPPER SOFT LIMIT POSITION

This command will set the Upper Soft Limit Position to the value given in the argument. Subsequent moves by the Move Absolute (**MA**), Move Relative (**MR**), Constant Velocity (**CV**) or manual Jog moves will not be allowed above this Upper Limit if the Soft Limits are enabled (see **SL** command).

Syntax	Units	Range	to	Initial State	Privilege level
<ad>ULnnn	Steps	Lower Limit	2147483647 ($\pm 2^{32}$)	2000000000	3

Condition Requirements	Notes:
Idle	Value stored in FLASH by BD command.

Responses

OK	Command has been accepted.
! LIMITS CONFLICT	Attempting to set upper limit below or equal to lower limit

Example:

1UL8000	Set the axis 1 Upper Soft Limit Position to 8000.

UP UNDEFINE PROFILE

This command will undefine or cancel a Profile definition. **Note** that this will only remove the profile definition from the volatile memory and to change the non-volatile flash memory this command must be followed by a backup profile (**BP**) command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>UP	Profile No.	0	7	N/A	5

Condition Requirements	Notes:
Servo mode, Idle.	Value stored in FLASH by BP command. Not used for stepper modes.

Responses:

OK	Command has been accepted.
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

1UP	Delete Profile from axis 1 controller.

US UNDEFINE SEQUENCE

This command will undefine or cancel a sequence definition. **Note** that this will only remove the sequence definition from the volatile memory and to change the non-volatile flash memory this command must be followed by a backup sequence (**BS**) command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>US	Sequence No.	0	7	N/A	5

Condition Requirements
Idle.

Notes:
Value stored in FLASH by BS command.

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument (sequence number) is out of valid range.

Example:

1US6	Delete sequence 6 from axis 1 controller.

WA WAIT FOR INPUT PORT CONDITION

This command will examine the read port inputs and compare them with the specified bit pattern argument. It will wait until the inputs are equal to the specified bit pattern before issuing its **'OK'** response and moving on to the next command.
The bit pattern is specified as a eight digit binary number of either **0, 1** or **2** characters starting with read port 8, through to 1. A **0** defines that the input must be low (0V or open-circuit), a **1** defines that the input must be high (+24V) and a **2** defines that the input is not relevant or 'don't care'. If less that 8 digits are specified in the argument, then the preceding ones are assumed as low (**0**).

Syntax	Units	Range	to	Initial State	Privilege level
<ad>WAbbbbbbbb	Bit pattern	8 digits of 0, 1 or 2		N/A	1

Condition Requirements
None.

Notes:

Responses

OK	Command has been accepted.
! INVALID BINARY	Invalid argument i.e. bit specified was not 0, 1 or 2 OR the number of bits was greater than 8.

Example:

1WA22112210 Will wait until the following condition is on the read input port before continuing:

PORT :	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
STATE :	(Ignored)	(Ignored)	High	High	(Ignored)	(Ignored)	High	Low

WE WAIT FOR END

This command will wait for the end of a move or delay. It will wait until any current move or delay has finished and detects the return to the *idle* state. The 'OK' response will not be issued until the move or delay has been completed. Therefore **WE** can be used to execute I/O commands after a move is complete.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>WE	N/A	N/A		N/A	1

Condition Requirements **Notes:**
Idle (by definition).

Response:

OK	Command has been completed.

Examples:

1MR4000	Move 4000 steps positive.
1WE	Wait for End of above move
1WP22222221	Turn LED on (write port 1) when move has finished.
1DE1000	Delay for 1 second.
1WE	Wait for End of Delay
1WP22222220	Turn LED off (write port 1).

WI SET WINDOW

This command will set the *window* for end of move checking. The *window* is the maximum acceptable difference between the *actual position* and the *command position* at the end of a move. The use of the 'end of move window' depends on the control mode.

CM1 Servo mode
At the end of a move, when the *actual position* comes within the **WI** range of this final target, the **SE** (settling time) counter counts down. When the settling time reaches zero the controller will either accept the next command or go to the *idle* condition.
If the Position overshoots the window before to the settling time reaches zero, the settling time counter is reset and started again. This means that for a move to be declared complete, the position difference (between *actual position* and *command position*) must be within the *window* for at least the *settling time*.

CM11 Open-loop Stepper mode
Not used.

CM12 Checking stepper mode
At the end of a move, the **SE** (settling time) counter counts down. After the settling time reaches zero, if the *actual position* is within the **WI** range, the controller will either accept the next command or go to the *idle* condition. If the *actual position* is outside the *window* a **! NOT COMPLETE/TIMEOUT ABORT** message will occur.

CM13 External loop stepper mode
The operation is the same as in described in servo mode.

CM14 Closed loop stepper mode
At the end of a move, the **SE** (settling time) counter counts down. When the settling time reaches zero and the *actual position* is within the **WI** range the controller and will either accept the next command or go to the *idle* condition. If the *actual position* is outside the *window* a then a correction move will take place.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>Winnn	Steps	0	2147483647 (2^{32})	4	6

Condition Requirements
Idle

Notes:
Value stored in FLASH by BD command.
Not used for open-loop stepper modes.

Responses

OK	Command has been accepted.
! OUT OF RANGE	Argument is out of valid range.

Example:

1WI2	Set the Window for axis 1 to 2 steps.

WP WRITE TO OUTPUT PORT

Write to output port. The PM600 controller has eight user output ports, known as write ports 1 to 8. This command will set the write port outputs to a state defined by the specified bit pattern argument. The bit pattern is specified as an eight digit binary number. The digits will be either **0, 1** or **2** characters starting with write port 8 through to 1

Format:	Eight digit binary string consisting of 0s, 1s or 2s.
	0 = *Off* 0V or open-circuit
	1 = *On* +24V (depending on the voltage of Write Port V_{source})
	2 = *Don't change*

Syntax	**Units**	**Range**	**to**	**Initial State**	**Privilege level**
<ad>WPbbbbbbbb	Bit pattern	8 digits of 0, 1 or 2		N/A	1

Condition Requirements
None.

Notes:
Initial state on power-up: all **0** = *Off*
The last *write* is shown on the **QA** page.

Responses:

OK	Command has been accepted.
! INVALID BINARY	Invalid argument i.e. bit specified was either not 0, 1 or 2 or the number of bits was greater than eight.

Example:
If a PM600 on axis 1 currently has the following states on its output write ports:

	PORT:	8	7	6	5	4	3	2	1
	STATE:	off	off	on	on	off	on	on	on
1WP12001200		Will set the outputs to:							
	PORT:	8	7	6	5	4	3	2	1
	STATE:	on	off	off	off	on	on	off	off
		1	2 (unchanged)	0	0	1	2 (unchanged)	0	0

WS WAIT FOR SYNCHRONISATION

This command will make the PM600 wait and not execute any more commands until the Input position equals the Motor Position. This command is used in Absolute gearbox mode.

Syntax	**Units**	**Range**	**to**	**Initial State**	**Privilege level**
<ad>WS	N/A	N/A		N/A	1

Condition Requirements
Synchronised in absolute gearbox.

Notes:
Not used for stepper modes.

Response:

OK	Command has been completed.

Example:

1GA	Axis 1 enter absolute gearbox mode.
1WS	Axis 1 wait for synchronisation.
1WP22222221	Axis 1 switch output ON

XC EXECUTE CAM

This command will execute the defined Cam profile. The argument selects which cam profile is to be executed (0 to 7). The cam profile must have already been defined with a Define Cam **DC** command.

Using the **XC** command the motor will only start to move when the *input position* (Encoder 3) divided by the *cam modulo* is equal to the equivalent *actual position* (Encoder 1) defined in the *cam* profile. The response to a **CO** command while the PM600 is waiting for the positions to synchronise is **01:Cam synchronisation**.

Important when using an absolute cam, the actual position must be within the range specified in the cam definition for synchronisation to be achieved.

Syntax	**Units**	**Range**	**to**	**Initial State**	**Privilege level**
<ad>XCnnn	Cam number	0	7	N/A	2

Condition Requirements
Servo mode, Idle

Notes:
Not used for stepper modes.

Responses

OK	Command has been accepted.
! HARD LIMIT	Hard limit for required direction is already activated
! SOFT LIMIT	Soft limit for required direction has already been reached
! CAM UNDEFINED	Cam profile hasn't yet been defined
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A timeout abort has been detected
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

1XC1	Axis 1, execute Cam number 1.

XP EXECUTE PROFILE

This command will execute the defined Profile. The argument selects which Profile is to be executed (0 to 7). The Profile must have already been defined with a Define Profile **DP** command. The move occurs at a rate, defined in milliseconds by the **PT** command, for each **MR** segment to be completed.

Syntax	**Units**	**Range**	**to**	**Initial State**	**Privilege level**
<ad>XPn	Profile No.	0	7	N/A	2

Condition Requirements
Servo mode, Idle

Notes:
Not used for stepper modes.

Responses:

OK	Command has been accepted.
! HARD LIMIT	Hard limit for required direction is already activated
! SOFT LIMIT	Soft limit for required direction has already been reached
! PROFILE UNDEFINED	Profile hasn't yet been defined
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A timeout abort has been detected
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

1PT100	Set profile time so that each segment takes 100 ms.
1XP5	Axis 1, execute Profile number 5.

XR EXECUTE RELATIVE CAM

This command will execute the defined Cam profile from the current *actual position*. The argument selects which cam profile is to be executed (0 to 7). The cam profile must have already been defined with a Define Cam **DC** command.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>XRnnn	Cam number	0	7	N/A	2

Condition Requirements	**Notes:**
Servo mode, Idle	Not used for stepper modes.

Responses

OK	Command has been accepted.
! HARD LIMIT	Hard limit for required direction is already activated
! SOFT LIMIT	Soft limit for required direction has already been reached
! CAM UNDEFINED	Cam profile hasn't yet been defined
! INPUT ABORT	An input abort has been detected
! STALL ABORT	A stall abort has been detected
! TRACKING ABORT	A tracking abort has been detected
! TIMEOUT ABORT	A timeout abort has been detected
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

Example:

1XR1	Axis 1, execute Cam number 1.

XS EXECUTE SEQUENCE

This command will start execution of a sequence. The argument selects which sequence is to be executed (0 to 7). The sequence must have already been defined with a Define Sequence **DS** command.
If the Execute Sequence (**XS**) command is encountered during a sequence, it will explicitly transfer control to the beginning of the sequence specified, whether it is the sequence already running or another sequence. It can therefore be used to make a loop type sequence or jump to any other sequence. Please note that it should not be considered as a subroutine. It is like a GOTO rather than a GOSUB.
A sequence execution can be stopped before completion, or if in a continuous loop, by a Control-C or ESCAPE command.

Control-C will stop any movement immediately, exit the sequence and return to idle.
Escape will decelerate any move to a stop, exit the sequence and return to idle.

Syntax	Units	Range	to	Initial State	Privilege level
<ad>XSn	Sequence No. 0	7	N/A	2	

Condition Requirements	**Notes:**
None.	

Responses:

! INVALID SEQUENCE NUMBER	Argument (sequence number) is out of valid range.
! SEQUENCE UNDEFINED	Sequence specified has not been defined yet.

Other responses may be generated by commands within the sequence. At the completion of the sequence, the response to the last command is sent.

Example:

1XS1	Execute sequence 1

XY CAM CO-ORDINATES

Set Cam co-ordinates. In Cam mode the slave motor is driven at a ratio of the Input encoder speed. This Cam profile is specified by two arguments separated by a **/** character.
The first point is always x=0, y=0. Co-ordinate pairs must be defined in order of increasing x co-ordinate.
The x co-ordinate of the last pair defines the *modulo*, that is the repeat distance. In the example given below the modulo is 750, so that the y values for x=200, x=950, x=1700, etc. are the same. Exit from *cam mode* can be achieved by either ESCAPE or ST commands.
To obtain the most accurate cam action the feedforward coefficient should be made equal to the velocity coefficient. KF=KV.
Cam positions are absolute, not relative, so that the motor position should be around zero before starting cam. The motor will only start to move when the *input* position divided by the *cam modulo* is equal to the equivalent motor position defined by the *cam*.

Syntax	**Units**	**Range**	**to**	**Initial State**	**Privilege level**
<ad>XYnnn/nnn					5
x-value	N/A	0	2147483647 (2^{32})	N/A	
y-value	N/A	-2147483647	2147483647 ($\pm 2^{32}$)	N/A	

Condition Requirements
Servo mode, Define Cam.

Notes:

Responses:

OK	Command has been accepted.
! OUT OF RANGE	Argument, either X or Y is out of valid range.
! CAM FULL	No memory space for further definition
!ILLEGAL CAM INSTRUCTION	Command can only be used in cam definition
! NOT MONOTONIC	All X values must increase during Cam Definition
! NOT ALLOWED IN STEPPER MODE	Only works in servo mode

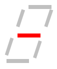
Example:
Cam profiles are *piecewise linear*, with the first co-ordinate implicitly (x=0, y=0). A cam profile would be defined using the following commands:

1DC	Open Cam definition.
1XY200/500	Second Cam co-ordinate.
1XY400/500	Next Cam co-ordinate.
1XY600/-200	Next Cam co-ordinate.
1XY700/-200	Next Cam co-ordinate.
1XY750/0	Last Cam co-ordinate.
1EM	End Cam Definition.

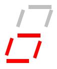
8 Error Messages

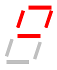
! BACKUP FAILURE	Unable to store parameters/sequence/cam/profile in flash
! CAM FULL	Define Cam memory is already full
! CAM UNDEFINED	Attempting to run Cam that hasn't yet been defined
! COMMAND ABORT	Cannot move when aborted by Command Abort
! CORRUPT BACKUP	Unable to get valid data from flash memory
! HARD LIMIT	Cannot move in direction of activated hard limit
! ILLEGAL ABORT MODE	Abort Mode parameters not correctly set
! ILLEGAL CAM INSTRUCTION	Cam Definition commands can only be XY and EC
! ILLEGAL INSTRUCTION	Instruction not recognised - probably typing mistake
! ILLEGAL PROFILE INSTRUCTION	Profile Definition commands can only be MR and EP
! ILLEGAL SEQUENCE INSTRUCTION	Command cannot be store in a Sequence Definition
! INPUT ABORT	Cannot move when aborted by Stop Abort Input
! INVALID BINARY	Binary expressions must be up to 8 characters of 0, 1 or 2
! INVALID CAM DEFINITION	Possibly due to no length Cam Definition
! INVALID CAM NUMBER	Only Cam Definitions 0 to 7 available
! INVALID MODE STRING	Attempting to set an invalid mode
! INVALID PIN	Attempting to enter wrong PIN number
! INVALID PROFILE DEFINITION	Possibly due to no length Profile Definition
! INVALID PROFILE NUMBER	Only Profile Definitions 0 to 7 available
! INVALID SEQUENCE NUMBER	Only Sequence Definitions 0 to 7 available
! JOYSTICK FAILURE	Joystick input out of range or not zero on power-up
! LIMITS CONFLICT	Attempting to set Soft Limits incorrectly
! NO VALID DATUM	There is no captured datum position to query or move to.
! NOT ABORTED	Attempting to reset a controller that is not aborted
! NOT ALLOWED IN STEPPER MODE	Attempting a servo only command
! NOT ALLOWED IN THIS MODE	Command not allowed whilst doing current command
! NOT COMPLETE/TIMEOUT ABORT	Cannot move whilst aborted; last move was not completed
! NOT MONOTONIC	All X values must increase during Cam Definition
! ONLY ALLOWED IN GBOX MODE	Attempting command that only functions in Gearbox mode
! OUT OF RANGE	Argument to command is outside allowable limits
! PRIVILEGE VIOLATION	Attempting command of higher privilege level than you have
! PROFILE FULL	Profile Definition is already full
! PROFILE UNDEFINED	Attempting to run Profile that hasn't yet been defined
! RS232 ABORT	Cannot move when aborted by receiving illegal character
! SEQUENCE FULL	Sequence Definition is already full
! SEQUENCE UNDEFINED	Attempting to run Sequence that hasn't yet been defined
! SOFT LIMIT	Attempting to move beyond a soft limit
! STALL ABORT	Cannot move when aborted by stall detect
! TRACKING ABORT	Cannot move when aborted by tracking error
! TUNE FAILURE	Auto Tune has not worked
Self-test failure	Power On Self Test detected failure
SKIPPED	Command was skipped due to preceding IT or IF command

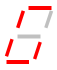
9 Status Display

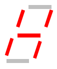

Idle

9.1 Moves

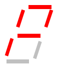

Move (MA, MR)

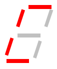

Constant velocity (CV)

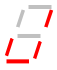

Gearbox (GA, GB)

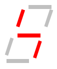

Home to datum (HD)

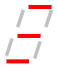
A dot shows in the PM600's status display when a datum position is captured.

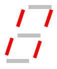

Profile running (XP)


Cam running (XC)

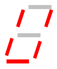

Jog or Joystick move

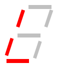

Stopping

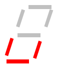

Tune (TUNE)

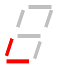

Wait for signal/condition (WA)

9.2 Limits

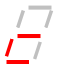
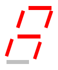
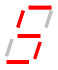
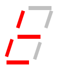
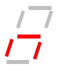

Upper hard-limit activated


Lower hard-limit activated

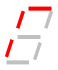
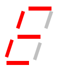
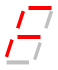

Upper soft-limit activated


Lower soft-limit activated

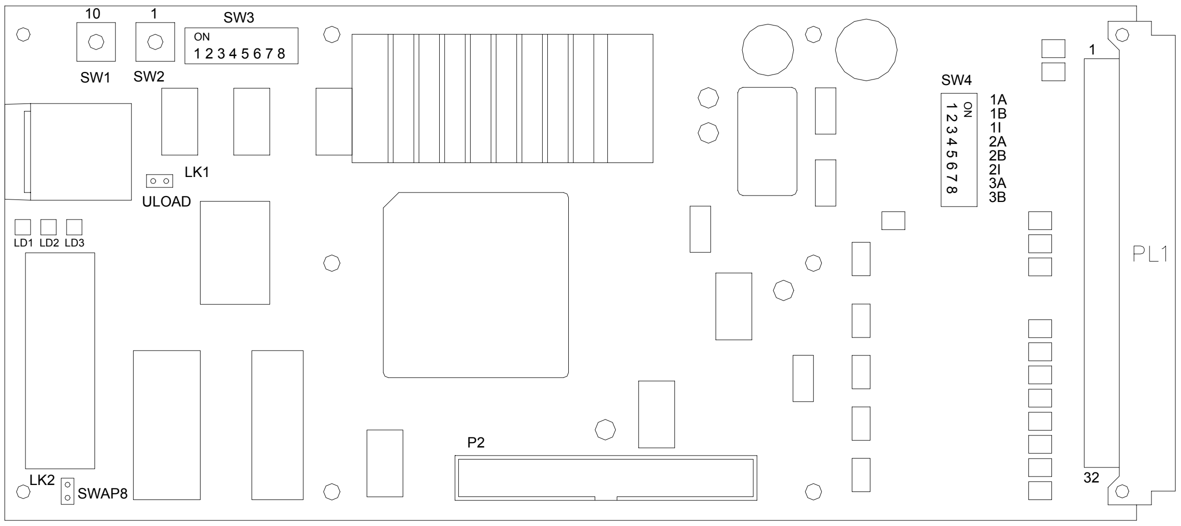
9.3 Aborts

	Command abort (**AB**)
	Stop Input Abort
	Stall abort (servo mode)
	Tracking error
	Time Out abort (not got there)

9.4 Errors

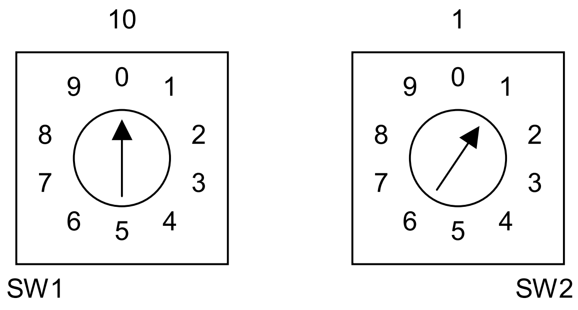
	Communication error
	Encoder quadrature error (Encoder 1 only)
	Controller failed self test

10 Switch Settings

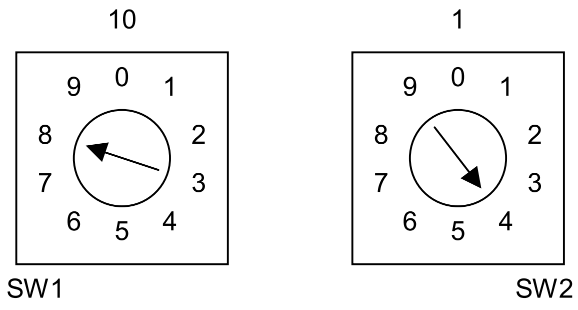


10.1 Axis Address Number SW1 & SW2

Rotary switches SW1 and SW2 are used to set the units axis address. This is the address of the serial commands that it will respond to. Typically, each controller in a system will be set to differing addresses. The left hand switch SW1 sets the decade value and SW2 sets the units. They can be set using a small screwdriver.



This example shows the address set to 1 (01).



This example shows the address set to 84.

10.2 Communication Configuration Switch SW3

DIP Switch SW3 is used to set the serial communication parameters. These should be set to match those of your host terminal or PC, and all PM600 units should be set the same. If the unit receives characters that do not match the set parameters it will cause a communication error abort.

Baud rate

1 ■□ ON 2 ■□	1 ■□ 2 ■□ 4800 baud	1 □■ 2 ■□ 9600 baud	1 ■□ 2 □■ 19200 baud	1 □■ 2 □■ 38400 baud

Word format

3 ■□	3 ■□ 7 bit even parity	3 □■ 8 bit no parity

Hardware handshake (RTS / CTS)

4 ■□	4 ■□ Disabled	4 □■ Enabled

Reply format

5 ■□	5 ■□ Verbose mode	5 □■ Quiet mode

Transmission format

6 ■□	6 ■□ Terminal mode	6 □■ Computer mode (Not Implemented – leave off)

Interface type used

7 ■□	7 ■□ RS232	7 □■ RS485

RS485 terminator select

8 ■□	8 ■□ No terminator	8 □■ 100Ω terminator

Reply format:

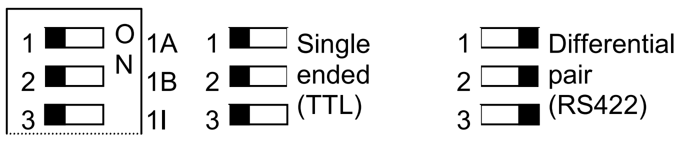
Verbose Mode -	Responses are given in full (e.g. **1OS** gives 01:Status = 00000000). Unsolicited responses are allowed (e.g. **01:! STALL ABORT**).
Quiet Mode -	Responses are given abridged (e.g. **1OS** gives 01:00000000). Unsolicited responses are NOT allowed (e.g. **01:! STALL ABORT**).

10.3 Encoder Termination Configuration Switch SW4

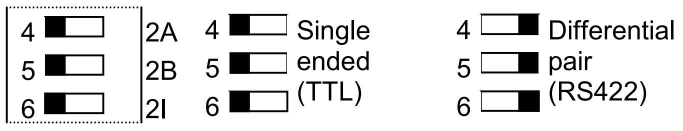
Switch SW4 selects the termination for the encoder signals. These can be set to OFF for open ended, TTL or open collector type encoder outputs or ON for 5V differential, RS422 type encoder outputs
There are switches for Channel-A, Channel-B and Index point inputs. The A and B switches should be set as a group for each encoder input

The correct termination should be set. This gives maximum noise immunity.

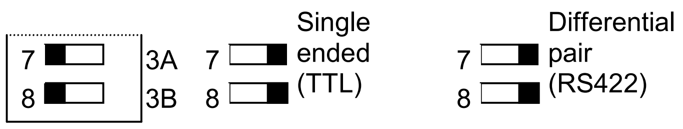
Encoder 1



Encoder 2



Encoder 3 (has no index input)



11 Electrical Specification

Signal	Pin no.	Characteristics	
+24 V Supply (+VLL)	1a,1b,1c	Supply voltage	+10.0 V to +32 V DC.
0V (0VLL)	32a,32b,32c	Supply current (With 250mA from +5V output)	420 mA @ 12 V 220 mA @ 24 V 190 mA @ 30 V
+5 V Output	5a,5b,5c	Output current	500 mA max.
Analogue Outputs	12a, 12b	Output voltage (load 2.2kΩ)	-10V to +10V nom.
		Resolution	-2047 to +2047 (12 bit)
Analogue 0v	13a, 13b		
Analogue Inputs	02c, 03c	Input voltage range	-10V (-2000) to +10V (+2000)
		Resolution	-2027 to +2048
		Input resistance	100KΩ
	04a, 04b	(Joystick inputs) Input voltage range	0V to +5V (+4095)
	04c	(Joystick centre tap) Input voltage range	0V to +5V (+4095)
		Resolution	0 to 4095
		Input resistance	3KΩ
Hard Limits/Datum	18a, 19a 20a, 21a	Opto-coupled inputs	
		Low level (0) input voltage	0 to +5V
		High level (1) input voltage	+10 to +35 V
		High level (1) input current	1mA min.
Isolated 0v	22a		
Abort Stop	10c	Opto-coupled inputs	
		Low level (0) input voltage	0 to +5V
		High level (1) input voltage	+10 to +35 V
		High level (1) input current	1mA min.
Isolated 0v	11c		
Jog +, Jog -	13c, 15c	Opto-coupled inputs	
Fast Jog	14c		
		Low level (0) input voltage	0 to +5V
		High level (1) input voltage	+10 to +35 V
		High level (1) input current	1mA min.
Isolated 0v	17c		
Read Ports	23b to 30b	Opto-coupled inputs	
		Low level (0) input voltage	0 to +5V
		High level (1) input voltage	+10 to +35 V
		High level (1) input current	1mA min.
Isolated 0v	31b		
Write Ports	23a to 30a	Opto-coupled outputs	
Voltage source	31a	Source Voltage	+35 V max.
		Output current	50mA max.
Enable	15a	Opto-coupled output	
Voltage source	14a	Source Voltage	+35 V max.
		Output current	50mA max.
Error	03b	Opto-coupled output	
Voltage source	02b	Source Voltage	+35 V max.
		Output current	50mA max.
Idle	03a	Opto-coupled output	
Voltage source	02a	Source Voltage	+35 V max.
		Output current	50mA max.
Stepper Outputs		Open-Collector referenced to 0VLL	
Step	16a	Pull up voltage	+35 V max.
Direction	17a	Sink current	50mA max.

Encoder Inputs:		Differential Inputs to RS422 Line receiver - 26LS32	
Encoder 1	06a to 11a	+ Input biased by 4K7Ω to +5V	
Encoder 2	06b to 11b	- Input biased to +5V by 15KΩ and 0V by 6K8Ω	
Encoder 3	06c to 09c	Termination switched by SW4, 1nF in series with 180Ω	
		5V TTL, open collector or single ended speed (SW4 off)	200K steps/sec max.
		5V Differential signals speed (SW4 on)	1.2M steps/sec max.
RS232		RS232 Compatible	
Transmit Output	18c	Output voltage swing into 3KΩ	±5V min. ±8V typ.
Receive Input	19c	Input threshold low	0.8V min. 1.2V typ.
RTS Output	20c	Input threshold high	1.7V typ. 2.4V max.
CTS Input	21c	Input resistance	3KΩ min. 5KΩ typ. 7KΩ max.
		Max. data rate	38400 bits/sec
RS485		RS485 Half-duplex compatible	
RS485-A	22c	Differential output (no load)	+5V max.
RS485-B	23c	Input voltage high	2.0V min.
		Input voltage low	0.8V max.
		Input resistance	12KΩ
		Common mode voltage	-7V to +12V
		Termination resistor (turned on by SW3-H)	100Ω