

ASCII Command Interface (ACI) Summary of Commands

Master Module Commands	
mmi n	Master initialize - resets all controllers and assign addresses 1 - <i>n</i>
mmeo	Turn off echo mode
mmwr	Write an 8-bit value to a register in the Motion Basic chip (this special command requires 2 parameters – an address followed by the value, separated by a space)
mm?n	Query number of modules connected
mm?r	Read an 8-bit value from a register in the Motion Basic chip
Generic Module Commands	
gm?t	Query module type (response: 0 = servo, 2 = I/O, 3 = stepper)
gm?v	Query module version number
PIC-SERVO Module Commands	
svin	Servo initialize
svcb	Clears latched status bits
svio	I/O control command (8-bit binary parameter)
svsa	Set acceleration (positive integer)
svsv	Set maximum velocity (positive integer)
svmp	Move to position (+/- integer)
svmv	Move at velocity (+/- integer)
svpw	Set PWM value, switch to PWM mode (-255 to +255 integer)
svzp	Zero position counter
svkp	Set gain Kp (0 – 32767 integer)
svkd	Set gain Kd (0 – 32767 integer)
svki	Set gain Ki (0 – 32767 integer)
svil	Set integration limit (0 – 32767 integer)
svol	Set output limit (0-255 integer)
svcl	Set current limit (0-255 integer)
svel	Set servo position error limit (0 – 32767 integer)
svsr	Set servo rate parameter (1-255 integer)
svdc	Set deadband compensation parameter (0-255 integer)
svsm	Set step multiplier (1-255 integer)
svhm	Set homing mode (8-bit binary parameter)
svss	Stop motor smoothly (decelerate to a stop)
svst	Stop motor abruptly (no deceleration)

svof	Turn off servo, disable amplifier
sv?s	Query module status byte
sv?p	Query motor position
sv?v	Query motor velocity
sv?a	Query motor A/D value / motor current
sv?x	Query motor auxiliary status byte (response: 8-bit binary value)
sv?h	Query home position
sv?e	Query servo position error
sv?d	Query move done (response: 0 = not done, 1 = done)
PIC-STEP Module Commands	
stin	Stepper initialize
stsa	Set acceleration parameter (1-250 integer)
stsv	Set maximum velocity (1-250 integer)
stmp	Move to position (+/- integer)
stmv	Move at velocity (+/- 1 - 250 integer)
stzp	Zero position counter
stsm	Set operating mode parameter (8-bit binary)
stlv	Set lowest velocity (1 – 250 integer)
strc	Set running current (0 – 255 integer)
sthc	Set holding current (0 - 255 integer)
sttl	Set thermal limit (0 - 255 integer)
stso	Set output bits (8-bit binary)
sthm	Set homing mode (8-bit binary)
stss	Stop motor smoothly (decelerate to a stop)
stst	Stop motor abruptly (no deceleration)
stof	Turn motor amplifier off
st?s	Query module status byte
st?p	Query motor position
st?a	Query A/D / temperature value
st?i	Query input bits (response: 8-bit binary value)
st?h	Query home position
st?d	Query move done (response: 0 = not done, 1 = done)

PIC-I/O Module Commands	
iodi	Make I/O bit 1 - 12 an input
iodo	Make I/O bit 1 - 12 an output
iosb	Set output bit 1 – 12 to a value of 1
iocb	Clear output bit 1 – 12 to a value of 0
iop1	Set PWM1 output value (0-255)
iop2	Set PWM2 output value (0-255)
ioct	Set counter/timer mode (8-bit binary)
io?s	Query module status byte
io?b	Query state of bit 1 - 12 (response: 0 or 1)
io?a	Query A/D input 1 – 3
io?c	Query counter/timer value

ASCII Command Interface (ACI) User Guide

The ASCII Command Interface (ACI) mode allows users to send command to **PIC-SERVO**, **PIC-STEP** and **PIC-I/O** controllers using simple ASCII commands sent from an RS232 terminal or using a Windows based terminal program like Hyperterminal. ACI mode is available for controller systems using our **Motion Basic** controller chip inserted into our **SSA-485** converter board.

The *native* communication mode for NMC controller modules is a binary format which is very efficient and powerful, but not always convenient. In ACI mode, however, the **Motion Basic** controller chip will accept simple ASCII command strings and translate them into the native binary format. For example: “1 svmp 1000” will move servo motor 1 to position 1000.

Please note: each ACI mode commands does not necessarily translate directly to a single native NMC command, and some more complex NMC commands are not available in ACI mode. While ACI commands are very easy to use, they are not as efficient as native NMC commands and may not be suitable for more complex applications.

Hardware Setup

Your NMC controller modules should be interconnected to the **SSA-485** board as indicated in the Quick Start section of the **SSA-485** board documentation (ssa485.pdf). The **Motion Basic** chip should be inserted in the 28-pin socket (labeled U4) on the **SSA-485** board. The two jumpers JP3 and JP4 in the **SSA-485** board should be moved to the “MB” positions. (On Version 1 of the **SSA-485** board, jumpers should be moved to the position marked “SSQ”).

If using RS232, connect your PC or other RS232 terminal device to the DB9 connector P1.

If using USB, connect to the USB connector on the **SSA-485** board. After turning on the logic power supply, use the Windows Control Panel (System->Hardware->Device Manager->Ports) to examine the properties of the USB serial port. We recommend setting the COM port number to COM5 or COM6 (hiding in the “Advanced” settings).

You should make sure that there is no **Motion Basic** program loaded in the EEPROM. This is the default state, but if you have been using the **Motion Basic** mode to write programs, use the Windows **Motion Basic** interface to erase any stored program. On power-up, the **Motion Basic / ACI** controller chip will determine that there is no program to run, and automatically enter ACI mode.

Running Hyperterminal

You can use the program Hyperterminal (or some other ASCII terminal program) to test the ACI mode. If running Hyperterminal, start by selecting File->Properties. In the “Connect To” tab, select “Direct to Comn” where *n* is the desired COM port number. Click on the “Configure” button and select 19200 baud, 8 data bits, No parity, 1 stop bit, and No flow control.

In the “Settings” tab, click on the ASCII Setup button. Select “Send line ends with line feeds” but do *not* select “Echo typed characters locally”.

When back at the terminal window, hit the “Enter” key a few times until you see a “:” character. You are now ready to send commands to the NMC controllers.

Basic Operation

Each ACI command begins with a module address, followed by a 4-character command, and optionally followed by a single numerical parameter. The command is executed as soon as the “Enter” key is pressed (*ie*, when an <new-line> character is sent). For example, in the command:

```
:1 svmp 1000<enter>
```

1 is the address of the target controller module. The command, “svmp”, stands “servo move position” and is the command used to move a servo controller to a specified position. The parameter 1000 is the goal position.

The ACI mode allows a maximum of 8 controller modules to be connected to one serial port and which are assigned addresses 1 – 8. Commands for the **Motion Basic**/ACI chip itself (the Master Module) use address 0.

All commands have 4 characters. The first 2 characters generally indicate the type of controller being used. For example, any command “sv--” will be a command for a **PIC-SERVO** type module. The table below shows the different types of module commands:

mm--	Master Module commands interpreted directly by the Motion Basic /ACI controller chip
gm--	Generic Module commands sent to individual controllers, but are not specific to any one type of controller
sv--	PIC-SERVO module commands
st--	PIC-STEP module commands
io--	PIC-I/O module commands

Master Module commands include commands to reset and initialize all connected controllers, and to query the number of controllers connected.

Generic Module commands are used for querying the module type and version for a particular address. This allows you to determine basic information about a module without having to know what type it is beforehand.

All remaining commands are commands specific to the **PIC-SERVO**, the **PIC-STEP** or the **PIC-I/O** controllers.

Special Characters

Almost all characters used in the ACI commands and replies are letters A-Z (either upper or lower case can be used), digits 0-9, the characters ‘?’, ‘!’ and ‘:’. The only additional characters used are the <new-line> (decimal value 10), the <carriage-return> (decimal value 13), and the

<backspace> (decimal value 8).

When sending a command, it should be terminated with a <new-line> character. The <carriage-return> character is optional, but if using a terminal program, it is useful to send a <carriage-return> along with the <new-line> for readability. (The default mode is that all characters are echoed back by the **Motion Basic** / ACI controller chip.)

When a query command is sent, the controller will send back a numeric value followed by a <new-line> and a <carriage-return>. This is followed by a ‘:’. After all non-query commands, just the ‘:’ will be sent.

Error Messages

The **Motion Basic** / ACI controller does some limited error checking of commands. If an error is detected in a command, a ‘!’ will be sent, followed immediately by a two-digit error code, followed by a text message describing the error. The text message will be terminated with a <new-line> and a <carriage-return>, and again followed by a ‘:’. A table of error messages appears at the end of this document.

A Simple Example

The following (hypothetical) example demonstrates using the ACI mode with one **PIC-SERVO** module and one **PIC-STEP** module. (Characters in **red** are those sent by the **Motion Basic**/ACI controller, characters in **blue** are those that you type.)

Start by typing:

```
:0 mmin<enter>
```

This command will send a reset command to all controllers and then assign each module a unique address. Next, type:

```
:0 mm?n  
2
```

to query the Master Module for the number of modules connected. It responds with a value of 2, followed by a <new-line> and a <carriage-return character>. (Note: from now on, <enter>, <new-line>, <carriage-return>, etc. will be omitted for clarity.)

To get the module type for module address 1, use the command:

```
:1 gm?t  
0
```

The response of 0 indicates a **PIC-SERVO** type module. Repeat for the next module:

```
:2 gm?t  
3
```

The response of 3 indicates a **PIC-STEP** module.

To make the **PIC-SERVO** controller actually do something, enter the “servo initialize” command:

```
:1 svin
```

This will do several things: 1) Set the servo gains to default values, 2) Enable the amplifier and enable the PID servo operation, 3) Clear the latched status bits. You will notice that your servo motor will now attempt to hold its current position. Finally, the command:

```
:1 svmp 1000
```

will move the motor to encoder position 1000 in trapezoidal profile mode with default acceleration and velocity values. To query the motor position, use the command:

```
:1 sv?p  
992
```

The reply of 992 indicates that the motor has moved to a position just short of the goal position by 8 encoder counts. This difference is the *servo position error*, and is typical in servo control systems.

Additional Documentation

This manual does not contain a complete description of **PIC-SERVO**, **PIC-STEP** and **PIC-I/O** controller modules. Please see the web page jrkerr.com/docs.html for the following additional documents:

PICSRCSV.PDF	Complete description of the operation of the PIC-SERVO SC controller chip.
PICSTEP.PDF	Complete description of the operation of the PIC-STEP controller chip.
PIODATA.PDF	Complete description of the operation of the PIC-I/O controller chip.

Command Reference

Master Module Commands

Command	<code>mmin</code>
Parameter	none
Response	none
Description	<u>Master Module Initialize</u> : Resets all controller modules and assigns them unique addresses. A maximum of 8 controller modules can be connected.
Example	<code>:0 mmin</code>
	Initialize all connected controller modules. Note: address 0 must be used with all Master Module commands.

Command	<code>mmeo</code>
Parameter	none
Response	none
Description	<u>Master Module Echo Off</u> : Turns off echoing of characters back to the terminal. This command may be useful when ACI commands are sent from a computer program rather than from a terminal.
Example	<code>:0 mmeo</code>
	Turn off echo mode. Note: address 0 must be used with all Master Module commands.

Command	<code>mmwr</code> – <i>for advanced applications</i>
Parameter	address, value
Response	none
Description	<u>Master Module Write Register</u> : Write to one of the hardware registers of the Motion Basic chip. This command give you access to unused hardware features of the Motion Basic chip.
Example	<code>:0 mmwr 3988 178</code>
	Writing a value of 178 to register 3988 (TRISC) turns pin RC2 of the Motion Basic chip into an output. Note that this special command has 2 parameters: the register address and the value to write into the register.

Command	mm?n
Parameter	none
Response	Number of modules connected (0 – 8)
Description	<u>Master Module Query Number of Modules</u> : Queries the Master Module for the number of controller modules connected.
Example	:0 mm?n 5
	The response indicates 5 controller modules are connected to the Master Module. Note: address 0 must be used with all Master Module commands.

Command	mm?r – <i>for advanced applications</i>
Parameter	address, value
Response	8-bit register value
Description	<u>Master Module Query Register</u> : Read one of the hardware registers of the Motion Basic chip. This command give you access to unused hardware features of the Motion Basic chip.
Example	:0 mm?r 3988 182
	Reads the value of register 3988 (TRISC).

Generic Module Commands

Command	gm?t
Parameter	none
Response	Module type: 0 = PIC-SERVO , 2 = PIC-I/O , 3 = PIC-STEP
Description	<u>Generic Module Query Type</u> : Queries a module for its type. This command is useful for determining what type of module is associated with each address.
Example	:1 gm?t 0
	The response of 0 indicates that module address 1 is a PIC-SERVO module.

Command	gm?v
Parameter	none
Response	Module version number
Description	<u>Generic Module Query Version</u> : Queries a module for its firmware version number.
Example	:1 gm?v 10
	The response of 10 firmware version number of module address 1.

PIC-SERVO Module Commands

Command	<code>svin</code>
Parameter	none
Response	none
Description	<p><u>Servo Initialize</u>: Initializes a PIC-SERVO module for operation:</p> <ol style="list-style-type: none"> 1. Sets gain parameters to the default values of: $K_p = 100$ $K_d = 1000$ $K_i = 0$ $IL = 2000$ $OL = 255$ $CL = 0$ $EL = 2000$ $SR = 1$ $DC = 1$ $SM = 1$ 2. Sets the Max. Velocity = 100,000 and the Acceleration = 100 3. Enables the motor amplifier and start the PID servo 4. Clears the latched status bits (OVERCURRENT, POS_ERROR, POS_WRAP, SERVO_OVERRUN).
Example	<code>:1 svin</code>
	Initializes PIC-SERVO module with address 1. Once this command is issued, the motor will attempt to hold its position when you try to rotate the shaft.

Command	<code>svcb</code>
Parameter	none
Response	none
Description	<p><u>Servo Clear Bits</u>: Clears the latched status bits: OVERCURRENT, POS_ERROR, POS_WRAP, SERVO_OVERRUN.</p>
Example	<code>:1 svcb</code>
	Clears latched status bits for PIC-SERVO module with address 1.

Command	svio
Parameter	<p>PIC-SERVO I/O control byte, sent as 8 binary digits (0 or 1):</p> <pre> 7 6 5 4 3 2 1 0 <- Bit number L-- Not used L---- Not used L----- Turn motor off if limit switch is HI L----- Stop motor abruptly if limit switch is HI L----- Use 3-phase commutation output mode L----- Use Antiphase PWM output mode L----- Not used L----- Use limit switch inputs for step & direction </pre>
Response	none
Description	<p><u>Servo I/O Mode</u>: Sets limit switch input options and PWM output options for the PIC-SERVO controller.</p> <p>By default, the limit switch inputs do not automatically stop the motor, and step & direction mode is disabled. To make the limit switches automatically stop the motor, set either bit 2 or bit 3 to '1'. To disable the limit switches altogether and use them as step & direction inputs instead, set bit 7 to '1', and make sure bits 2 and 3 are '0'. (Note: limit switch input values can be examined directly with the 'sv?s' command.)</p> <p>The default output mode is PWM and Direction mode. Your amplifier, however, may require the use of 3-phase or Antiphase output modes. Note that in most systems, this parameter has already been stored in EEPROM and set to the proper value on power-up. If, however, you use this command to change the behavior of the limit switch inputs, you must also make sure bits 4 and 5 are set to the proper value. (Bits 4 and 5 should never both be set.)</p>
Example	<pre>:1 svio 00010100</pre> <p>Setting bit 2 will cause the motor to turn off if a limit switch is hit. Setting bit 4 enables 3-phase commutation mode for the amplifier.</p>

Command	svsa
Parameter	Acceleration parameter (positive integer)
Response	none
Description	<p><u>Servo Set Acceleration</u>: The acceleration is in units of encoder counts per servo tick per servo tick, and then multiplied by 2^{16} to increase the resolution. (1 servo tick = 0.000512 seconds.) If your desired acceleration is in rev/sec/sec, the acceleration parameter is calculated as:</p> $\text{accel. parameter} = (\text{accel in rev/sec/sec}) \times (\text{encoder counts/rev}) \times 0.0172$ <p>The acceleration value is used in position mode to accelerate the motor up to speed also to decelerate the motor to a stop at the goal position. It is used in velocity mode when accelerating or decelerating from one speed to another.</p>
Example	:1 svsa 69
	For a motor with 2000 encoder counts per rev (<i>i.e.</i> , a 500 line encoder), this sets the acceleration to about 2 rev/sec/sec..

Command	svsv
Parameter	Velocity parameter (positive integer)
Response	none
Description	<p><u>Servo Set Velocity</u>: The maximum velocity is in units of encoder counts per servo tick, and then multiplied by 2^{16} to increase the resolution. (1 servo tick = 0.000512 seconds.) If your desired velocity is in rev/sec, the velocity parameter is calculated as:</p> $\text{vel. parameter} = (\text{vel in rev/sec}) \times (\text{encoder counts/rev}) \times 33.55$ <p>Note that this velocity is used as the maximum allowable velocity when moving in position mode.</p>
Example	:1 svsv 671000
	For a motor with 2000 encoder counts per rev (<i>i.e.</i> , a 500 line encoder), this sets the maximum velocity to about 10 rev/sec/sec..

Command	svmp
Parameter	Position value (+/- integer)
Response	none
Description	<p><u>Servo Move to Position</u>: Moves to a position (specified as the number of encoder counts) by accelerating up to the maximum allowable velocity, slewing at that velocity, and then decelerating to a stop at the goal position. See commands 'svsa' and 'svsv' for setting the acceleration and velocity.</p>
Example	:1 svmp -2000
	For a motor with 2000 encoder counts per rev (<i>i.e.</i> , a 500 line encoder), this moves the motor in the reverse direction by one revolution.

Command	svmv
Parameter	Velocity value (+/- integer)
Response	none
Description	<p><u>Servo Move at Velocity</u>: Accelerates or decelerates to the specified velocity. The velocity value is in units of encoder counts per servo tick, and then multiplied by 2^{16} to increase the resolution. (1 servo tick = 0.000512 seconds.) If your desired velocity is in rev/sec, the velocity is calculated as: $\text{vel. parameter} = (\text{vel in rev/sec}) \times (\text{encoder counts/rev}) \times 33.55$</p> <p>See the command 'svsa' for setting the acceleration.</p>
Example	:1 svmv -671000
	For a motor with 2000 encoder counts per rev (<i>i.e.</i> , a 500 line encoder), this runs the motor in the reverse direction at a speed of about 10 rev/sec.

Command	svpw
Parameter	PWM value (-255 to +255)
Response	none
Description	<p><u>Servo PWM mode</u>: Operates the motor in raw PWM output mode. In this mode, the speed of the motor will fluctuate depending on the load applied to the motor. The voltage applied to the motor is proportional to the PWM value. +255 corresponds to full forward voltage and -255 corresponds to full reverse voltage.</p>
Example	:1 svpw 128
	Drive the motor forward at half-voltage.

Command	svzpz
Parameter	none
Response	none
Description	<p><u>Servo Zero Position</u>: Resets the motor position to zero. Note that all position commands and position values reported back are absolute rather than relative. This command can be used to zero the position the counter to a useful operating point after homing.</p>
Example	:1 svzpz
	Set the motor position to zero.

Command	svkp
Parameter	Position gain Kp (integer 0 – 32,767)
Response	none
Description	<u>Servo set Kp</u> : Sets the position gain value Kp. Please refer to Appendix B for more details on setting gain values.
Example	:1 svkp 800
	Sets Kp to a value of 800.

Command	svkd
Parameter	Derivative gain Kd (integer 0 – 32,767)
Response	none
Description	<u>Servo set Kd</u> : Sets the derivative gain value Kd. Please refer to Appendix B for more details on setting gain values.
Example	:1 svkd 2000
	Sets Kd to a value of 2000.

Command	svki
Parameter	Integral gain Ki (integer 0 – 32,767)
Response	none
Description	<u>Servo set Ki</u> : Sets the integral gain value Ki. Please refer to Appendix B for more details on setting gain values.
Example	:1 svkd 50
	Sets Ki to a value of 50.

Command	svil
Parameter	Integration limit (integer 0 – 32,767)
Response	none
Description	<u>Servo set IL</u> : Sets the integral limit value IL. Please refer to Appendix B for more details on setting gain values.
Example	:1 svil 1000
	Sets IL to a value of 1000.

Command	svol
Parameter	Output limit (integer 0 – 255)
Response	none
Description	<u>Servo set OL</u> : Sets the output limit value OL. Please refer to Appendix B for more details on setting gain values.
Example	:1 svol 200
	Sets OL to a value of 200.

Command	svcl
Parameter	Current limit (integer 0 – 255)
Response	none
Description	<u>Servo set CL</u> : Sets the current limit value CL. Note that not all systems may use current sensing, and this parameter may be set to 0. Please refer to Appendix B for more details on setting gain values.
Example	:1 svcl 25
	Sets CL to a value of 25.

Command	svel
Parameter	Error limit (integer 0 – 32,767)
Response	none
Description	<u>Servo set EL</u> : Sets the error limit value EL. Please refer to Appendix B for more details on setting gain values.
Example	:1 svel 2000
	Sets EL to a value of 2000.

Command	svsr
Parameter	Servo Rate Parameter (integer 1-255)
Response	none
Description	<u>Servo set SR</u> : Sets the servo rate parameter SR. Please refer to Appendix B for more details on setting gain values.
Example	:1 svsr 2
	Sets SR to a value of 2.

Command	svdc
Parameter	Deadband Compensation Parameter (integer 1-255)
Response	none
Description	<u>Servo set DC</u> : Sets the deadband compensation parameter DC. Please refer to Appendix B for more details on setting gain values.
Example	:1 svdc 1
	Sets DC to a value of 1.

Command	svsm
Parameter	Step Multiplier (integer 1-255)
Response	none
Description	<u>Servo set SM</u> : Sets the step multiplier SM. SM is equal to the number of encoder counts commanded per step pulse.
Example	:1 svsm 15
	Sets SM to a value of 15 encoder counts per step pulse.

Command	svhm
Parameter	<p>Homing mode control byte sent as 8 binary digits (0 or 1):</p> <pre> 7 6 5 4 3 2 1 0 <- Bit number L-- On Limit 1 L---- On Limit 2 L----- Motor Off L----- On Index L----- Stop Abrupt (no deceleration) L----- Stop Smooth (decelerate to a stop) L----- On Position Error L----- On Overcurrent </pre>
Response	none
Description	<p><u>Servo set Homing Mode</u>: Sets the homing mode control byte. Bits 0, 1, 3, 6 and 7 specify which conditions will be used for the homing trigger. Homing will be triggered when <i>any</i> of the specified homing conditions <i>change</i> state. <i>e.g.</i>, if Limit 1 starts HI, homing will be triggered if it goes LO, or if it starts LO, homing will be triggered when it goes HI.</p> <p>When homing is triggered, the current position of the motor will be stored in the PIC-SERVO's home position register. If one of bits 2, 4 or 5 is set, the motor will also stop automatically.</p> <p>Note that this command does not initiate any motion. After you issue a homing command, you should then issue a motion command to move the motor towards one of the homing triggers.</p>
Example	<pre>:1 svhm 00100011</pre> <p>Setting bits 0, 1 and 5 will cause homing to be triggered when either Limit 1 or Limit 2 change state (HI->LO or LO->HI), and the motor will stop smoothly.</p>

Command	svss
Parameter	none
Response	none
Description	<u>Servo Stop Smoothly</u> : Decelerates the motor to a stop.
Example	<pre>:1 svss</pre> <p>Servo motor address 1 decelerates to a stop.</p>

Command	svst
Parameter	none
Response	none
Description	<u>Servo Stop</u> : Stop motor abruptly with no deceleration.
Example	:1 svst
	Servo motor address 1 jerks to a stop.

Command	svof
Parameter	none
Response	none
Description	<u>Servo Off</u> : Turns off the PID servo and disables the amplifier
Example	:1 svof
	Turns off servo motor address 1.

Command	sv?s
Parameter	none
Response	<p>PIC-SERVO Module status byte returned as 8 binary digits (0 or 1):</p> <pre> 7 6 5 4 3 2 1 0 <- Bit number L-- Move Done L---- Communications Error L----- Over current L----- Power On L----- Position Error L----- Limit Switch 1 L----- Limit Switch 2 L----- Homing in Progress </pre>
Description	<u>Servo Query Status</u> : Queries the servo module for its current status byte.
Example	:1 sv?s 00101001
	The HI bits 0, 3 and 5 in the response indicate the current move is done, the motor power supply is ON, and Limit Switch 1 is HI.

Command	sv?p
Parameter	none
Response	Motor position in encoder counts (+/- integer)
Description	<u>Servo Query Position</u> : Queries the servo module for the current motor position.
Example	:1 sv?p -21376
	The value returned is the actual position of the motor reported in encoder counts.

Command	sv?v
Parameter	none
Response	Motor velocity in encoder counts per servo tick (0.000512 sec.). (+/- integer)
Description	<u>Servo Query Velocity</u> : Queries the servo module for the current motor velocity.
Example	:1 sv?v 12
	The value returned is the motor velocity. Note that, unlike the commanded velocity value, the velocity value returned is not scaled by 2 ¹⁶ . (See command 'svmv').

Command	sv?a
Parameter	none
Response	PIC-SERVO A/D (motor current) conversion value (0 – 255)
Description	<u>Servo Query A/D</u> : Queries the servo module for the A/D value. If the PIC-SERVO controller is configured for current sensing, this value is proportional to the motor current.
Example	:1 sv?a 58
	The value returned is the A/D conversion value. If using a PIC-SERVO SC controller board current sensing (which produces 39 A/D counts per amp), a reading of 58 would equal about 1.5 amps.

Command	sv?x
Parameter	none
Response	<p>PIC-SERVO Module auxiliary status byte returned as 8 binary digits (0 or 1):</p> <pre> 7 6 5 4 3 2 1 0 <- Bit number L-- Encoder Index L---- Position wrap (32 bit position counter wrap around) L----- Servo On (PID servo is active) L----- Accelerating (1) / Decelerating (0) (Only valid if not slewing) L----- Slewing (Motor is slewing at constant speed) L----- Servo overrun (Internal timing overrun) L----- Path mode (Path mode is still active) L----- Not used </pre>
Description	<u>Servo Query Auxiliary Status</u> : Queries the servo module for its current auxiliary status byte.
Example	<pre>:1 sv?x 00011100</pre> <p>The HI bits 2, 3 and 4 in the response indicate the PID servo is active, and the motor is slewing at a constant speed. Note: bit 3 is ignored if bit 4 is active.</p>

Command	sv?h
Parameter	none
Response	Motor home position in encoder counts (+/- integer)
Description	<u>Servo Query Home Position</u> : Queries the servo module for the value stored in the PIC-SERVO 's home position register. This is the position of the motor when the home condition was triggered. (See 'svhm' command).
Example	<pre>:1 sv?h 1428</pre> <p>If homing was set to trigger on the encoder index signal, the value reported is the motor position where the index signal occurred.</p>

Command	sv?e
Parameter	none
Response	Servo position error (+/- integer)
Description	<u>Servo Query Error</u> : Queries the servo module for the current servo position error. This error is the difference between the commanded motor position and the actual motor position.
Example	<pre>:1 sv?e -14</pre> <p>The value of -14 indicates that the motor is <i>ahead</i> of the commanded position by 14 encoder counts.</p>

Command	sv?d
Parameter	none
Response	'1' if move is done, '0' if not complete
Description	<u>Servo Query Done</u> : Queries the servo module if the current move is done. If the most recent motion command is a position command, move done means the motor has reached its goal position. If a velocity mode command, it means the motor has reached the command velocity. Note that if the PID servo is not active, a value of '1' will be returned by default.
Example	:1 sv?d 1
	If a position command has been issued, '1' indicates that the motor has reached the goal position.

PIC-STEP Module Commands

Command	stin
Parameter	none
Response	none
Description	<p><u>Stepper Initialize</u>: Initializes a PIC-STEP module for operation:</p> <ol style="list-style-type: none"> 1. Sets operating parameters to the default values of: Speed mode = 1x, Limit switches ignored, E-stop ignored, Min. speed = 1 Run current = 20, Hold current = 10, Therm. limit = 0 2. Sets the Max. Velocity = 20 and the Acceleration parameter = 15 3. Sets output bits 1, 2 and 3 (These are the recommended values when using the KAE-T3V1-BDV1 PIC-STEP board.) 4. Enables the amplifier.
Example	:1 stin
	Initializes PIC-STEP module with address 1. Once this command is issued, the motor will hold its position when you try to rotate the shaft.

Command	stsa
Parameter	Acceleration parameter (positive integer 1 - 250)
Response	none
Description	<p><u>Stepper Set Acceleration Parameter</u>: The acceleration parameter is the amount of time spent at one speed before incrementing to the next higher or lower speed (while accelerating or decelerating). Therefore, a higher acceleration parameter will result in a lower acceleration. The acceleration parameter is in units of 0.25 millisecc. The time to accelerate from one speed to another is given by: $\text{accel. Time} = (0.00025 \times \text{Accel Param}) \times (\text{Final speed} - \text{Starting Speed})$ </p> <p>The acceleration value is used in position mode to accelerate the motor up to speed also to decelerate the motor to a stop at the goal position. It is used in velocity mode when accelerating or decelerating from one speed to another.</p>
Example	:1 stsa 20
	Sets the acceleration parameter to 20. For example, if you were to start at a speed to 10 and accelerate to a speed of 150, it would take 0.70 seconds.

Command	stsv
Parameter	Velocity parameter (positive integer 1 - 250)
Response	none
Description	<p><u>Stepper Set Maximum Velocity:</u> The velocity parameter is an integer value between 1 and 250. The exact step rate depends on the speed mode set with the “stsm” command. In 1x speed mode, each increment of the velocity parameter corresponds to 25 steps per second. In 2x mode, it equals 50 steps per sec., in 4x mode, it equals 100 steps per second, and in 8x mode, it equals 200 steps per second. Therefore, the overall range of speeds goes from 25 step/sec (1x mode, vel = 1) to 50,000 step/sec (8x mode, vel = 250).</p> <p>Note: the step rate governs the number of step pulses per second. If you are using half-stepping or microstepping, the step rate is actually the number of half steps or microsteps per second.</p> <p>The velocity value is used in trapezoidal position mode as the maximum slewing velocity. Note that for short moves with slow accelerations, the maximum velocity may never be reached.</p>
Example	:1 stsa 100
	Sets the maximum velocity to 100. If you were using 2x speed mode, this would correspond to 5000 steps per second..

Command	stmp
Parameter	Position value (+/- integer)
Response	none
Description	<p><u>Stepper Move Position:</u> Moves to the specified position with a trapezoidal profile. (<i>i.e.</i>, The motor accelerates up to the maximum velocity, slews, and then decelerates to a stop at the goal position.) The position is specified in steps. If your amplifier is using half-steps or microsteps, the position will be specified in half-steps or microsteps. Note that position values are absolute.</p>
Example	:1 stmp 400
	Moves the motor to a position of 400 steps.

Command	stmv
Parameter	Velocity value (1 to 250, -1 to -250)
Response	none
Description	<p><u>Stepper Move Velocity</u>: Moves the motor at the specified velocity, accelerating or decelerating as needed. Note that 0 is not a valid velocity parameter. Also note that if you are moving at a positive velocity, you cannot issue a command to move at a negative velocity without first using a stepper stop motor command (“stss” or “stst”) to stop the motor first. (The same applies for going from a negative to a positive velocity.)</p> <p>The velocity parameter is an integer value between 1 and 250 (or between -1 and -250). The exact step rate depends on the speed mode set with the “stsm” command. In 1x speed mode, each increment of the velocity parameter corresponds to 25 steps per second. In 2x mode, it equals 50 steps per sec., in 4x mode, it equals 100 steps per second, and in 8x mode, it equals 200 steps per second. Therefore, the overall range of speeds goes from 25 step/sec (1x mode, vel = 1) to 50,000 step/sec (8x mode, vel = 250). Note: the step rate governs the number of step pulses per second. If you are using half-stepping or microstepping, the step rate is actually the number of half-steps or microsteps per second.</p>
Example	:1 stmp 400
	Moves the motor to a position of 400 steps.

Command	stzp
Parameter	none
Response	none
Description	<p><u>Stepper Zero Position</u>: Resets the motor position to zero. Note that all position commands and position values reported back are absolute rather than relative. This command can be used to zero the position the counter to a useful operating point after homing.</p>
Example	:1 stzp
	Set the motor position to zero.

Command	stsm
Parameter	<p>Operating mode byte sent as 8 binary digits (0 or 1):</p> <pre> 7 6 5 4 3 2 1 0 <- Bit number L L-- Speed mode bits: 11 = 1x, 10 = 2x, 01 = 4x, 00 = 8x L----- Ignore the limit switches L----- Ignore the E-stop L----- Turn amplifier off on Limit or E-stop L----- Not used, set to 0 L----- Not used, set to 0 L----- Not used, set to 0 </pre>
Response	none
Description	<u>Stepper Set Mode</u> : Sets the operating mode parameter byte. The mode byte specifies the speed mode, and the behavior of the limit switches and the e-stop switch.
Example	<pre>:1 stsm 00010011</pre> <p>Setting bits 0 and 1 specifies the 1x speed mode. Setting bit 4 will disable the amplifier if a limit switch or the e-stop switch is hit.</p>

Command	stlv
Parameter	Minimum operating velocity (1 – 250)
Response	none
Description	<u>Stepper set Lowest Velocity</u> : When running at very low speeds, stepper motors can hit resonances which disrupt the smooth operation of the motor. To avoid this, a minimum operating velocity can be set. If a stopped motor is commanded to move to a new speed, the speed will first jump up to the minimum operating velocity, and then continue accelerating up to the desired velocity. When decelerating to a stop, the speed will be lowered down to the minimum operating velocity and then jump zero speed.
Example	<pre>:1 stlv 4</pre> <p>If in 1x speed mode, this will set the minimum operating velocity to 100 steps/sec..</p>

Command	strc
Parameter	Motor running current (0 – 255)
Response	none
Description	<u>Stepper set Running Current</u> : Sets running current output level. The running current controls the amount of current used while a motor is moving. A value of 255 corresponds to the maximum available amplifier current, a value of 0 corresponds to near zero output. (If using the KAE-T3V1-BDV1 PIC-STEP board, this value should be set no higher than 200.)
Example	:1 strc 100
	Sets the running current to about 40% of the maximum.

Command	sthc
Parameter	Motor holding current (0 – 255)
Response	none
Description	<u>Stepper set Holding Current</u> : Sets holding current output level. The holding current controls the amount of current used while a motor is stopped and holding its position. A value of 255 corresponds to the maximum available amplifier current, a value of 0 corresponds to near zero output. (If using the KAE-T3V1-BDV1 PIC-STEP board, this value should be set no higher than 200.)
Example	:1 sthc 50
	Sets the holding current to about 20% of the maximum.

Command	sttl
Parameter	Thermal limit (0 – 255)
Response	none
Description	<u>Stepper set Thermal Limit</u> : Sets the thermal limit parameter. If the TEMP_SENS input of the PIC-STEP is connected to a thermistor bridge which lowers its voltage as the temperature rises, this parameter can be used to automatically disable the amplifier as the temperature rises. A value of 255 corresponds to a 5.0v input and 0 corresponds to a 0.0v input.
Example	:1 sttl 128
	Sets the thermal limit parameter to 128. If the voltage on the TEMP_SENS input drops below 2.5v, the amplifier will be disabled.

Command	stso
Parameter	<p>Output bits 0 - 4 sent as 8 binary digits (0 or 1):</p> <pre> 7 6 5 4 3 2 1 0 <- Bit number L-- Output bit 0 L---- Output bit 1 L----- Output bit 2 L----- Output bit 3 L----- Output bit 4 L----- Not used, set to 0 L----- Not used, set to 0 L----- Not used, set to 0 </pre>
Response	none
Description	<u>Stepper set Outputs</u> : Sets auxiliary output bits 0 – 4. If using the KAE-T3V1-BDV1 PIC-STEP board, bits 1, 2 and 3 should be set for normal half-step operation.
Example	:1 stso 00001110 Sets output bits 1, 2 and 3 to 1. Bits 0 and 4 are set to 0.

Command	sthm
Parameter	<p>Homing mode control byte sent as 8 binary digits (0 or 1):</p> <pre> 7 6 5 4 3 2 1 0 <- Bit number L-- On Limit 1 L---- On Limit 2 L----- Motor Off L----- On Home input L----- Stop Abrupt (no deceleration) L----- Stop Smooth (decelerate to a stop) L----- Not used, set to 0 L----- Not used, set to 0 </pre>
Response	none
Description	<p><u>Stepper set Homing Mode</u>: Sets the homing mode control byte. Bits 0, 1, and 3 specify which conditions will be used for the homing trigger. Homing will be triggered when <i>any</i> of the specified homing conditions <i>change</i> state. <i>e.g.</i>, if Limit 1 starts HI, homing will be triggered if it goes LO, or if it starts LO, homing will be triggered when it goes HI.</p> <p>When homing is triggered, the current position of the motor will be stored in the PIC-STEP's home position register. If one of bits 2, 4 or 5 is set, the motor will also stop automatically.</p> <p>Note that this command does not initiate any motion. After you issue a homing command, you should then issue a motion command to move the motor towards one of the homing triggers.</p>
Example	<pre>:1 sthm 00100011</pre> <p>Setting bits 0, 1 and 5 will cause homing to be triggered when either Limit 1 or Limit 2 change state (HI->LO or LO->HI), and the motor will stop smoothly.</p>

Command	stss
Parameter	none
Response	none
Description	<u>Stepper Stop Smooth</u> : Decelerates the motor to a stop.
Example	<pre>:1 stss</pre> <p>Decelerated the motor to a stop.</p>

Command	stst
Parameter	none
Response	none
Description	<u>Stepper Stop Motor</u> : Stops motor abruptly. This command can also be used for re-enabling the amplifier after a “stof” command.
Example	:1 stst
	Stops the motor abruptly.

Command	stof
Parameter	none
Response	none
Description	<u>Stepper Off</u> : Stops the motor and disables the motor amplifier. The motor will no longer hold its position.
Example	:1 stof
	Turns off the motor amplifier.

Command	st?s
Parameter	none
Response	<p>PIC-STEP Module status byte returned as 8 binary digits (0 or 1):</p> <pre> 7 6 5 4 3 2 1 0 <- Bit number L--- Motor Moving L---- Communications Error L----- Amplifier Enabled L----- Power On L----- Motor At Commanded Speed L----- Controller in Velocity Mode L----- Controller in Trapezoidal Position Mode L----- Homing in Progress </pre>
Description	<u>Stepper Query Status</u> : Queries the stepper module for its current status byte.
Example	:1 st?s 01001101
	The HI bits 0, 2, 3 and 6 in the response indicate the motor is still moving, the amplifier is enabled, motor power supply is ON, and the controller is in trapezoidal profile mode.

Command	st?p
Parameter	none
Response	Motor position in steps (+/- integer)
Description	<u>Stepper Query Position</u> : Queries the stepper module for the current motor position.
Example	:1 st?p -832
	The value returned is the actual position of the motor reported in steps.

Command	st?a
Parameter	none
Response	PIC-STEP A/D (temp. sense) conversion value (0 – 255)
Description	<u>Stepper Query A/D</u> : Queries the stepper module for the A/D value. If the PIC-STEP controller is configured for temperature sensing, this value is inversely proportional to the thermistor temperature.
Example	:1 st?a 58
	The value returned is the A/D conversion value.

Command	st?i
Parameter	none
Response	<p>PIC-STEP input bits returned as 8 binary digits (0 or 1):</p> <pre> 7 6 5 4 3 2 1 0 <- Bit number L-- E-Stop input L---- General purpose input 1 L----- General purpose input 2 L----- Limit switch 1 L----- Limit switch 2 L----- Home Switch input L----- Not used - returned as 0 L----- Not used - returned as 0 </pre>
Description	<u>Stepper Query Inputs</u> : Queries the stepper module for its current input byte.
Example	:1 st?i 00001000
	The HI bit 3 indicates the Limit switch 1 is HI, all other inputs are LO.

Command	st?h
Parameter	none
Response	Home position register value in steps (+/- integer)
Description	<u>Stepper Query Home position</u> : Queries the stepper module for the value in the Home Position Register. This register holds the motor position for when the homing condition was triggered. See the “sthm” command for details on setting the homing conditions.
Example	:1 st?h 212
	The value returned is the position where the home condition was triggered.

Command	st?d
Parameter	none
Response	‘1’ if move is done, ‘0’ if not complete
Description	<u>Stepper Query Done</u> : Queries the stepper module if the current move is done. If the most recent motion command is a position command, move done means the motor has reached its goal position. If a velocity mode command, it means the motor has reached the command velocity.
Example	:1 st?d 1
	If a position command has been issued, ‘1’ indicates that the motor has reached the goal position.

PIC-I/O Module Commands

Command	iodi
Parameter	Bit number (1 – 12)
Response	none
Description	<u>I/O Digital Input</u> : Makes the specified digital I/O pin an input (rather than an output). On power-up, all digital I/O pins default to inputs.
Example	:1 iodi 3
	Makes the PIC-I/O digital I/O bit 3 and input.

Command	iodo
Parameter	Bit number (1 – 12)
Response	none
Description	<u>I/O Digital Output</u> : Makes the specified digital I/O pin an output (rather than an input). The value of this output (HI or LO) can be set with the commands “iosb” or “iocb”. Caution should be used to make sure that the pin specified is not connected to another output pin.
Example	:1 iodo 4
	Makes the PIC-I/O digital I/O bit 4 and output.

Command	iosb
Parameter	Bit number (1 – 12)
Response	none
Description	<u>I/O Set Output</u> : Sets a bit previously defined as an output to a HI state.
Example	:1 iosb 4
	Set bit 4 to a HI state.

Command	iocb
Parameter	Bit number (1 – 12)
Response	none
Description	<u>I/O Clear Output</u> : Clears a bit previously defined as an output to a LO state.
Example	:1 iocb 4
	Clears bit 4 to a LO state.

Command	iop1
Parameter	PWM value (0-255)
Response	none
Description	<u>I/O set PWM1</u> : Sets the PWM1 output to the specified value. A value of 255 corresponds to a 100% duty cycle PWM signal, and a value of 0 corresponds to a 0% signal..
Example	:1 iop1 128
	Sets PWM1 to a 50% duty cycle.

Command	iop2
Parameter	PWM value (0-255)
Response	none
Description	<u>I/O set PWM2</u> : Sets the PWM2 output to the specified value. A value of 255 corresponds to a 100% duty cycle PWM signal, and a value of 0 corresponds to a 0% signal..
Example	:1 iop2 64
	Sets PWM2 to a 25% duty cycle.

Command	ioct
Parameter	<p>Counter/timer mode byte sent as 8 binary digits (0 or 1):</p> <pre> 7 6 5 4 3 2 1 0 <- Bit number L-- Enable counter/timer L---- 1 for Counter mode, 0 for Timer mode L----- Not used, set to 0 L----- Not used, set to 0 L L----- Scaling factor bits: 00 counts every event, 01 = ÷ by 2, 10 = ÷ by 4, 11 = ÷ by 8 L----- Not used, set to 0 L----- Not used, set to 0 </pre>
Response	none
Description	<p><u>I/O set Counter / Timer mode</u>: Sets the operating mode for the PIC-I/O Counter/Timer. In counter mode, it can be configured to count every rising signal edge on the counter/timer input pin, every 2nd edge, every 4th edge or every 8th edge.</p> <p>In timer mode, the PIC-I/O's internal 5MHz timer is connected to the counter. Therefore, it can be configured to increment at a rate of 5MHz, 2.5 Mhz, 1.25MHz, or 0.625MHz.</p> <p>Every time "ioct" is used, the counter/timer value will be reset to 0.</p>
Example	<pre>:1 ioct 00110001</pre> <p>Setting bits 0, 4 and 5 will enable the counter/timer in timer mode, counting at a rate of 0.625MHz. The counter/timer value will also be cleared.</p>

Command	io?s
Parameter	none
Response	<p>PIC-I/O Module status byte returned as 8 binary digits (0 or 1):</p> <pre> 7 6 5 4 3 2 1 0 <- Bit number L-- Not used L---- Communications Error L----- Not used L----- Not used L L----- Not used L----- Not used L----- Not used L----- Not used </pre>
Description	<u>I/O Query Status</u> : Queries the I/O module for its current status byte.
Example	<pre>:1 io?s 00000000</pre> <p>There is no communications error.</p>

Command	io?b
Parameter	Bit number (1 – 12)
Response	'0' or '1'
Description	<u>I/O Query Bit</u> : Reads the value of the specified bit. The bit may be defined as either an input or an output.
Example	:1 io?b 3 1
	Indicates bit 3 is HI.

Command	io?a
Parameter	A/D channel number (1, 2 or 3)
Response	Positive integer (0 - 255)
Description	<u>I/O Query A/D</u> : Reads the value of the specified analog input channel. A response of 0 corresponds to 0.0v, and a response of 255 corresponds to +5.0v.
Example	:1 io?a 1 64
	Analog input 1 is equal to 1.25v.

Command	io?c
Parameter	none
Response	Positive integer
Description	<u>I/O Query Counter value</u> : Reads the value of the counter/timer. The meaning of the response will depend on the mode set with the "ioct" command.
Example	:1 io?c 156250
	If the counter timer was configured at a timer counting in divide by 8 mode, the value of 156250 would equal 0.25 seconds.

Appendix A – Error Codes and Messages

The following errors may be generated when issuing commands:

- !01 Syntax error – Invalid command syntax
- !02 Address error – Invalid address specified
- !03 Module type error – Module is of the wrong type for the command
- !08 Communications error – Error in communicating with the controller module
- !09 Too many modules connected – More than 8 modules connected
- !10 Unknown command – Invalid command specified

Appendix B – PIC-SERVO PID Control Parameters

In general, when in position or velocity mode, the motor is controlled by a servo loop which once every servo tick (1953.125 times/sec) looks at the current position of the motor, compares it to where the motor should be (the command position), and then uses a “PID control filter” to calculate an output which will cause the difference in positions, or the “position error” to become smaller. How well the motor follows the command position is dependent on a set of control filter parameters.

The control filter used by the **PIC-SERVO** is a “proportional-integral-derivative”, or PID filter. The output to the motor amplifier is the sum of three components: one proportional to the position error providing most of the error correction, one proportional the *change* in the position error which provides a stabilizing damping effect, and one proportional to the accumulated position error which helps to cancel out any long-term error, or “steady state” error.

The PID control filter, operating on the command position and the actual position each servo tick, produces an output calculated as follows:

$$output = Kp \times pos_error - Kd \times (pos_error - prev_pos_error) + Ki \times integral_error$$

The term *pos_error* is simply the current command position minus the actual position. The *prev_pos_error* is the position error from the previous servo tick. *Kp*, *Ki* and *Kd* are the 16 bit servo gains which will be programmed to optimize performance for your particular motor.

The *integral_error* is the running sum of *pos_error* divided by 256. To keep from growing a potentially huge *integral_error*, the running sum is bounded by a 16 bit user specified integration limit, *IL*. (Note that by temporarily setting the integration limit to 0, the user can zero out the accumulated running sum.)

The actual PWM output value (0-255) and direction bit are given by:

$$PWM = \min[\text{abs}(output/256) + dead_band , output_limit] - current_limit_adjustment$$
$$DIR = 0 \text{ if } output > 0, \quad DIR = 1 \text{ if } output < 0$$

The parameter *dead_band* is an 8-bit offset used to compensate for static friction or a dead band region in the amplifier.

First note that the scaled *PWM* output is limited by an 8-bit user defined *output_limit*. For example, if you are using a 12v motor powered by 24v, you would want to set the *output_limit* to 255/2, or 127. Also note that the final PWM value is reduced by a *current_limit_adjustment*. This value is explained in Section 4.7 below.

The *PWM* signal is a 19.53 KHz square wave of varying duty cycle with a *PWM* value of 255 corresponding to 100% and a value of 0 corresponding to 0%. As explained here, the *PWM* signal is derived from an 8-bit value. The internal calculations, however, actually provide and additional 2 bits of resolution for a final 10 bit *PWM* resolution.

An additional control parameter is the user specified 16 bit *position error limit*. If abs (*pos_error*) becomes larger than this limit, the position servo will be disabled. This is useful for disabling the servo automatically upon a collision or stall condition. Also, the POS_ERROR bit in the status byte will be set on a position error condition. (This condition can also be used for homing the motor by intentionally running it up against a limit stop.)

Selection of the optimal PID control parameters can be done analytically, but more typically, they are chosen through experimentation. As a first cut, the following procedure may be used:

1. First set the position gain, Kp , and the integral gain, Ki , to 0. Keep increasing the derivative gain, Kd , until the motor starts to hum, and then back off a little bit. The motor shaft should feel more sluggish as the value for Kd is increased.
2. With Kd set at this maximal value, start increasing Kp and commanding test motions until the motor starts to overshoot the goal, then back off a little. Test motions should be small motions with very large acceleration and velocity. This will cause the trapezoidal profiling to jump to goal position in a single tick, giving the true step response of the motor.
3. Depending on the dynamics of your system, the motor may have a steady state error with Kp and Kd set as above. If this is the case, first set a value for the integration limit IL of 16000 and then start increasing the value of Ki until the steady state error is reduced to an acceptable level within an acceptable time. Increasing Ki will typically introduce some overshoot in the position. The best value for Kp will be some compromise between overshoot and settling time.
4. Finally, reduce the value of IL to the minimum value which will still cancel out any steady state error.

There is one final servo filter parameter to discuss. The servo calculations above are executed at a rate of 1.953 KHz. For systems with a combination of a large inertia, little inherent damping and limited encoder resolution, it may be difficult to get sufficient damping at low speeds because the digitization noise with very large values of Kd will cause the servo to hum or vibrate. To decrease the digitization noise, the *servo rate divisor* (SRD) parameter is used to calculate how many servo cycles elapse between calculating the *pos_error* and the *prev_pos_error* used in the damping term of the PID filter. By default, $SRD = 1$, and *prev_pos_error* is equal to the *pos_error* of one cycle earlier. If, however, we set $SRD = 3$, *prev_pos_error* would equal the *pos_error* of three cycles earlier. Increasing the time difference between these values effectively averages the derivative error term and reduces the digitization noise.

In summary, we have a total of eight control filter parameters: Position Gain (Kp), Derivative Gain (Kd), Integral Gain (Ki), Integration Limit (IL), Output Limit (OL), Current Limit (CL), Position Error Limit (EL) and the Servo Rate Divisor (SRD). When using ACI mode, each of these parameters is set with a separate command..