ANIMATICS

Page Number:	Page 1 of 34
Revision:	H
Effective:	May 5, 2001
Replaces:	November 16, 1999

SmartMotor™ and RTC3000 Controller
Expanded Capabilities List

Motion Control Chip Firmware Version :
4.00-Q1 to 4.00-Q3 ,
4.10 to 4.15,4.75
4.40 to 4.41

Animatics is upgrading SmartMotors™ with its new firmware Version 4.15,4.75,4.40 & 4.41 Motion Control Firmware. This new version is backward compatible with your older programs, but has substantially increased functionality.

This document is intended to serve as an addendum to the existing manual until a new manual can be published.

A CD containing the Windows based terminal software, SmartMotor Interface "SMI", should come with the manual. The software is also downloadable on our website at www.animatics.com or www.smartmotor.com. For first time users, refer to the Help section in SmartMotor Interface (SMI) software.

To find out your SmartMotor firmware version, type the following command in the "SmartMotor Terminal" window "**RSP**" and the motor responds by returning the sample rate and firmware version. For example a response of "**24576/412**" the numbers after the slash "/" is the firmware version number "412" means you have a 4.12 firmware version motor. A "xxxxxx/Qx" means you have a version 4.00-Qx motor (where x are numbers). A "xxxxx-Mx" means you have a version 3.4 motor.

TABLE OF CONTENTS

INTRODUCTION

This section covers features and motion commands that are not covered in the current SmartMotor Manual.

The SmartMotor is a brushless DC servo-motor that uses a built in controller and amplifier to perform programmed motion. The servo controller uses closed loop PID control. The SmartMotor and RTC3000 can either be run from a program downloaded to its memory or from commands received from RS232 or RS485 communication cable.

We recommend first time users to program the SmartMotor and RTC3000 in the Windows™ based SmartMotor Interface (SMI) software. SmartMotors can also run on the DOS based interface software TermV400. Both software comes on the CD with the SmartMotor and is also available on the website at:
http://www.animatics.com/support.shtml

Click on SMIsetup.exe (single file) or SMIdisk1.exe to SMIdisk4.exe to save onto floppy disks.

The SmartMotor and RTC3000 can be to receive commands from other software that sends ASCII characters such as C++,C,Visual Basic , data acquisition software and industrial devices.

For first time users refer to help-topics in the Windows based SmartMotor Interface Software (SMI). To get to these files open the SMI software and select Help>Helptopics from the toolbar. The help topics include material covered in the printed manual, additional program examples, hardware diagrams and the full command sets of the different firmware versions.

When starting the SMI and TermV400 software for the first time it defaults to com port 1, 9600 baud, RS232 protocol. Please change these settings for your system com port. The SmartMotor controller and RT3000 defaults to 9600 baud, RS232, 8data bits, 1stop bit, non-parity through the 7 pin connector.

FIRWARE VERSIONS:

Firmware version 4.15:

Differences from 4.12:

- Added Directional Limit and Limit Trigger High/Low software selectable function. (LIMD,LIMN,LIMH and LIML)
  Default is non-directional limit (LIMN) and limit trigger low (LIML).
- Added Coordinated Motion (Multi-axis contouring)
- The motor will not auto-execute the stored program on the EEPROM if the characters "EE" is transmitted to the motor within ½ second after power up.
- ADDR=<expression> used for setting motor address (SADDR<value> is also accepted) on dasiy chain/multidrop . (SADDR<value> is also accepted)
- F=8 command added. Command causes the PID integral term to be cleared at the end of the trajectory.
- MD50 command added. This puts the motor into torque mode and sets the "T" by reading analog voltage at port A. The "T" value uses a linear scale where:
  5.0v input sets T=1023
  2.5v sets T=0,
  0v sets T=-1023

Firmware version 4.75:

Differences from 4.15:

- Default is Directional limit (LIMD) and limit trigger high (LIMH).

Firmware version 4.40/4.41:

Differences from 4.12:

- Added Direction Limit and Limit Trigger High/Low software selectable function. (LIMD,LIMN,LIMH and LIML)
  Default is non-directional limit (LIMN) and limit trigger low (LIML).
- Added Coordinated Motion (Multi-axis contouring)
- No External Encoder Inputs (ENCA input & ENCB input)
- No Mode Follow, Mode Follow with ratio and Mode CAM.
- The motor will not auto-execute the stored program on the EEPROM if the characters "EE" is transmitted to the motor within ½ second after power up.
- ADDR=<expression> used for setting motor address on RS232 daisy chain/ RS485 multi-drop . (SADDR<value> is also accepted)

SECTION 1.0: CREATING MOTION

The main modes of operation of the SmartMotor and RTC3000 controller:
1) Position Mode-User wants to control position of motor.
Motor moves to set position in at user defined acceleration and velocity.
SmartMotors default to this mode on power-up, unless otherwise defined in the program downloaded to program memory.

2) Velocity Mode-User wants to have a constant controlled velocity.
Motor rotates at a user set acceleration and constant velocity.

3) Torque Mode-User wants to control torque and speed of motor.

To decelerate to a stop issue "X". To stop as fast as possible issue "S" command.

POSITION MODE:
Position Mode controls the position of the shaft based on encoder signal. The SmartMotor defaults to reads its differential internal encoder (section 1.3 covers external encoder implementation).
SmartMotors come standard with :
500 line encoder for SM1720,SM23xx
1000 line encoder in the SM34xx,SM42xx,SM56xx
The encoder lines are read in quadrature (the 4 edges of encoder lines are read) that allows the motors to be controlled to a resolution of 2000 encoder counts (steps) per revolution for a 500 line encoder (0.18 degree per encoder count. And 4000 encoder counts (steps) per revolution for a 1000 line encoder (0.09 degree per encoder count).

In this mode you want to get to a position **"P"** at a desired acceleration **"A"** and desired velocity **"V"**. The controller accelerates to the set velocity and decelerates (at same rate of acceleration) to a stop at set position. Note commands are in uppercase and variable names are in lower case letters.

Position Mode can be used as absolute position **"P"** (move relative zero position) or in relative postion move **"D"** (move relative to current shaft position). Refer to Section 1.1 to calculate "A" and "V" values.
Note a positive velocity value is a clockwise rotation if you are looking at the mounting plate and shaft. Negative is counterclockwise. Position uses same sign convention. Note SmartMotor commands are shown in bold and a different font.

Sample:

```
MP          ' set to position mode
O=0         ' set current shaft position as  zero position called Origin or Home
            '  (command is letter O equal zero)
A=100       'set acceleration
V=32212     'set velocity
            Absolute Position Move
P=2000      ' move to position 2000 in the positive direction relative to the zero position.
G           'go and start move

            Relative Position Move
D=-8000     ' move 8000 encoder counts in negative direction relative to current shaft position.
G           'go and start move
```

VELOCITY MODE:

Velocity mode is used when a controlled constant velocity is desired. In this mode you have to set the velocity and acceleration. Note a positive velocity value is a clockwise rotation if you are looking at the mounting plate and shaft. Negative is counterclockwise.

Note that in this mode you can make on-the-fly (real-time) velocity changes, the new velocity is initiated when the go command **"G"** is issued.

Example:

```
MV          'set to mode velocity, clockwise rotation.
A=80        ' set acceleration
V=32212*5   ' set velocity
G           'go and start motion

V=32212*20  'change desired velocity
A=200       'change acceleration
G           ‘go and start this new velocity and new acceleration.

V=-50000    'change desired velocity, counter clockwise. This accelerates at the last acceleration value that
             ' was set.
G           ‘go and start this new velocity and current acceleration.
```

TORQUE MODE

In this mode the SmartMotor controls the amount of power to the motor. In this mode only issue values of "T", do not issue a "G" this will cause the motor to exit torque mode . The "T" is a torque factor the range is from 0 to 1023. Where 1023 is maximum power (speed and torque) available from the power supply.

Note that Animatics torque speed curves are rated at 48V DC. To get rated speed and torque use Animatics p/n PS42V6A, 42V unregulated power supply or equivalent. Maximum speed will decrease if using a lower voltage power supply. Note the “AMPS” command will not set the current limit. In torque mode the power to the motor is controlled by “T” value to limit the torque. Position error limit has no effect in torque mode , it will not cause the motor to stop.

```
MT          'set to mode torque
T=50        ' set to about 5% max power available (positive direction)
T=-500      'change to about 50% max power available (negative direction)
T=1023      'set to maximum power available (speed and torque)
T=0         'set to zero speed and torque
```

SECTION 1.1: COMPUTING VELOCITY (V), ACCELERATION (A) and WAIT values

To achieve a better level of control and improved performance over the speed range the SmartMotor internally scales the servo-samples. The controller uses velocity in units of scaled counts per second and acceleration in units of scaled counts per second squared. We have provided equations and sample calculations that convert standard units (Rpm, rps and rev/s^2) to the SmartMotor units in section 1.1. Section 1.1a covers firmware versions 4.10 to 4.15, 4.75,4.40 to4.41. Firmware version 4.00-Q1 to 4.00-Q3 are covered in section 1.1b covers 4.00-Q1 to4.00-Q3. Please refer to label on motor for firmware version or send the command "RSP" to the motor.

The SmartMotor's servo-controller takes samples the encoder position at a sample rate of 4069 times per second (version 4.00 and above) .

SECTION 1.1a: COMPUTING V, A, and WAIT values for FirmwareVersion 4.10 to 4.15,4.75, 4.40 to 4.41

Standard Units: Angular Velocity: Rpm = revolution/minute or Rps = revolution/ second
Angular Acceleration: revolution/ second^2

SmartMotor Units: velocity [**V**] = scaled encoder counts/sec
acceleration [**A**] = scaled encoder counts/sec^2

The Following are the equations to convert from standard units to SmartMotor **V**elocity & **A**cceleration values

`V=<expression>` 'where the expression can be an integer, variable
`A=<expression> '` or one mathematical operation.

Refer to Animatics product selection chart to determine standard encoder resolution.

For a 2000 encoder count per revolution:

1 revolution=131,072,000 scaled counts
V = (# rpm)*(536.87633 scaled encoder counts per sec/rpm)
or V = (# rps)*(32212.578 scaled encoder counts per sec/ rps)
A = (# rev/s^2)*(7.9166433 scaled encoder counts per sec^2/ rev per sec^2)

For a 4000 encoder count per revolution:

1 revolution=262,141,000 scaled counts
V = (# rpm)*(1073.7526 scaled encoder counts per sec/rpm)
or V = (# rps)*(64425.156 scaled encoder counts per sec/ rps)
A = (1 rev / s^2)*(15.833286 scaled encoder counts per sec^2/ rev per sec^2)

Sample Rate = 24576.25 seconds / 100,000,000 servo samples = 1 second / 4068.9695 servo samples .

`WAIT=` #servo samples For example: **`WAIT=4069`** is 1 second

SAMPLE CALCULATION:

If you want: velocity of 1 revolution/second (equal to 60 rpm) and an acceleration of 1 rev/s^2 set "V" and "A".

2000 count encoder: (SM17xx, SM23xx)
Your SmartMotor (**`V= #`**) and (**`A=#`**) are:

V=(60 rpm)*536.87633=32212.58 **`V=60*537`**
Or V=(1 rev/sec)*32212.578=32212.58 **`V=32213`**
A=(1 rev/s^2)*7.9166433=7.92 **`A=8`**

SAMPLE CALCULATION:
If you want: velocity of 1 revolution/second (equal to 60 rpm) and an acceleration of 1 rev/s^2 set "V" and "A".

4000 count encoder (SM34xx, SM42xx, SM56xx)
Your SmartMotor V= <expression> and A=< expression > are:

	V=(60 rpm)*1073.7526=644425.16	**V=64425**
Or	V=(1 rev/sec)* 64425.156 =64425.16	**V=64425**
	A=(1 rev/s^2)* 15.833286=15.83	**A=16**

SECTION 1.1b: COMPUTING V, A, and WAIT values for FirmwareVersion 4.00-Q1 to 4.00-Q3

Standard Units: Angular Velocity: Rpm = revolution/minute
or Rps = revolution/ second
Angular Acceleration: revolution/ second^2
SmartMotor Units: Velocity [**V**] = scaled encoder counts/sec
Acceleration [**A**] = scaled encoder counts/sec^2

For SM17xx series SmartMotor™

Sample Rate = 24576.25 seconds / 100,000,000 servo samples = 1 second / 4068.9695 servo samples .

WAIT= #servo samples
For example: **WAIT=4069** is 1 second

The following are equations to convert from standard units to SmartMotor **V**elocity & **A**cceleration values.
Velocity and Acceleration values are defined as: **V=expression**
A=expression

V = (# rpm)*(536.87633 scaled encoder counts per sec/rpm)
or V = (# rps)*(32212.578 scaled encoder counts per sec/ rps)
A = (# rev/s^2)*(7.9166433 scaled encoder counts per sec^2/ rev per sec^2)

SAMPLE CALCULATION:
If you want: velocity of 1 revolution/second (equal to 60 rpm) and an acceleration of 1 rev/s^2 .

2000 count encoder:
1 rev=131,072,000 scaled encoder counts
Your SmartMotor (**V=** and (**A=#**) are:

	V=(60 rpm)*536.87633=32212.58	**V=32213**
or	V=(1 rev/sec)*32212.578=32212.58	**V=32213**
	A=(1 rev/s^2)*7.9166433=7.92	**A=8**

For SM23xx and SM34xx series SmartMotor™

Sample Rate = 24824.24 seconds / 100,000,000 servo samples . 1 second / 4028.3207 servo samples .

WAIT= #servo samples

For example: **WAIT=4069** is 1 second

2000 count encoder:

1 rev = 131,072,000 scaled encoder counts

V = (# rpm)*(542.29383 scaled encoder counts per sec/rpm)

or V = (# rps)*(32537.628 scaled encoder counts per sec/ rps)

A = (# rev/s^2)*(8.07722 scaled encoder counts per sec^2/ rev per sec^2)

4000 count encoder:

1 rev = 262,144,000 scaled encoder counts

V = (# rpm)*(1084.587596 scaled encoder counts per sec/rpm)

or V = (# rps)*(65075.226 scaled encoder counts per sec/ rps)

A = (# rev/s^2)*(16.15444 scaled encoder counts per sec^2/ rev per sec^2)

SAMPLE CALCULATION : SM23xx and SM34xx series version 4.00-Qx:

If you want: velocity of 1 revolution/second (equal to 60 rpm) and an acceleration of 1 rev/s^2 .

2000 count encoder:

Your SmartMotor **V** and **A** values are:

V=(60 rpm)* 542.29383 =32537.63 **V=32538**

or V=(1 rev/sec)* 32537.628 =32537.63 **V=32538**

A=(1 rev/s^2)* 8.07722 =8.08 **A=8**

4000 count encoder:

Your SmartMotor V= <expression> and A=< expression > are:

V=(60 rpm)* 1084.587596=65075.23 **V=65075**

or V=(1 rev/sec)* 65075.226 =65075.23 **V=65075**

A=(1 rev/s^2)* 15.833286=15.83 **A=16**

SECTION 1.2: INTEGRAL BRAKE COMMANDS

For motors equipped with Animatics integral brakes, a series of new commands and dedicated internal I/O facilitate the use of the integral brake on version 4.12 and newer.

BRKENG	Engage the brake.
BRKRLS	Release the brake.
BRKSRV	Engage the brake whenever the motor is not servo-ing.
BRKTRJ	Brake on and servo off, if no trajectory

SECTION 1.3: EXTERNAL ENCODER COMMANDS

Please refer to the Animatics Product Selection Chart or Pinout drawing to see if your motor has the ability to accept external encoder inputs. To use a differential external encoder input you have to setup the hardware by wiring encoder A signal to the ENCA pin (port A) and encoder B signal to ENCB pin (port B). To allow the controller to use the external encoder as the primary encoder, in place of the internal encoder , issue the "ENC1" command from the interface software or in the motor's program memory.

Encoder commands:

ENC0 Encoder zero: (Default) Use internal encoder as primary encoder

ENC1 Encoder one: Use external encoder as primary encoder and the shaft position is recorded in the counter (CTR)

SECTION 1.4: Vertical Axis Applications , Gravity Constant (KG, KGON and KGOFF commands)

COMMAND DESCRIPTION

KGON This command reduces torque ripple and smoothes motion against an externally applied constant force on the axis. An example of such a force is gravity acting on a lead screw driven vertical axis slide. At low speeds, the peak torque increases, but the continuous torque decreases. The stability also changes so the PID filter coefficients should be modified.

KG=<value> A positive value of KG is in the positive direction (clockwise looking at the shaft) and applies a constant amount of power to the motor to compensate for the constant applied force.

KGOFF Turns off the KGON function.

To use the KGON issue following commands in this sequence:

```
KG=<value>   'sets amount of power to motor to compensate for constant
             'applied force.
F            'Update the PID (F)ilter.
KGON         'Turn on the KG function.
```

SECTION 1.5: Directional and Non-directional Limit inputs (Left/Right Limit Switches)

Port D and C is set to Left Limit and Right Limit Input functions by default. The limits switches can be set as directional or non-directional on firmware version 4.15, 4.75, 4.40 and 4.41.

On version 4.15,4.40 and 4.41 the default is non-directional limit "LIMN" and limit active low "LIML" and can be changed by issuing LIMD and LIMH.

On version 4.75 the default is directional limit "LIMD" and limit active high "LIMH".

Directional Limits Version:

Firmware version 4.15, 4.75, 4.40 and 4.41, have the directional limits capability. To use directional limits obtain SmartMotor Interface (SMI) Software version 1.221 or newer (available from Animatic's tech support).

Non-Directional Limits Version:

Firmware version (4.13 and lower, 4.00-Qx, 3.00) have non-directional limits and can not use the LIMH,LIML,LIMD and LIMN commands. For example, if the left limit input (port D) is triggered the motor will stop. However if you issue a move and "G" going left (Clockwise, looking at the shaft) the motor will go left.

Directional commands:(firmware version 4.15, 4.75, 4.40 and 4.41)

LIMN LIMits are Non-directional, as they have been before. This is the default.

LIMD LIMits are Directional.

A new occurrence of either limit halts the motor. A move begun on a limit is only allowed to move in the opposite direction of the limit.

If the left limit (counter clockwise) is triggered, the motor is not allowed to go past that left limit point.

If the right limit (clockwise) is triggered, the motor is not allowed to go past that right limit point.

Limit Trigger Low/High Commands: (versions 4.15, 4.40 and newer)

LIML LIMit active Low triggers when signal goes from low input. This is the default.

LIMH LIMit active high, sets limit input to trigger on high input

SECTION 1.6: MOTOR and LOAD PROTECTION FEATURES

The SmartMotorTM servo motor is equipped with several protection features and diagnostic tools that allow the user to protect and perform diagnostic functions on the load. These features are: power limit, temperature , error and power monitoring functions.

SECTION 1.6a: Error Limit

The allowable position error limit is software settable by issuing **E=<value>**. The range of the values is 0 to 32000 (default is **E=1000**). When the position error reaches the error limit "**E**" the motor stops execution of the program, stops the shaft and turns off the amplifier and the error limit status bit "**Be**" goes to 1. To reset the error limit bit issue a go **"G"**.

This affects the following commands: **E=**, **RE**, **=E**, **RPE**, and **=@PE**.
This error limit range is useful in compensating for large position errors while using the VRE (Variable Resolution Encoder).

If error limit is achieved in applications where motor is run at a constant high speed, increase/decrease the PID filter velocity constant "**KV**". The constant compensates for the non-changing error value associated in high speed operations. Issue this sequence of commands.

KV=<value> 'set constant velocity error PID filter constant.

F 'update PID Filter.

SECTION 1.6b: Peak Power Limit

The internal peak power limit of the SmartMotor™ servomotor is set by the AMPS command. This function controls amount of power that is delivered to the motor. Note, the AMPS command limits output torque and maximum speed, as well.
The valid range of the "AMPS" command is a scale factor with range 0 to 1023. The default setting is 1000.

For example, a setting of **`AMPS=512`** will limit the output stall torque to ½ of the peak torque rating and limit the maximum velocity to ½ of the specification. To get full torque and speed set the value, AMPS=1023.

`AMPS` can be assigned to a variable. For example, **`i=AMPS`** will store the value of AMPS in the variable i.

SECTION 1.6c: Temperature and RMS (Continuous) Power Limits

The RMS, continuous, power consumption is constantly monitored by the SmartMotorTM servo motor. If the RMS power exceeds continuous output rating of the SmartMotorTM servo motor for a programmable amount of time, the amplifier will shut down and indicate an overheat error (see status bit Bh). This programmable time is set by the THD (Temperature Hot, time Delay) function.

The valid range for THD is 0 through 65535 (version 4.40 THD range: 0-19967), with units in servo samples. For example, `THD=4069` will set the thermal shut down delay to one second. This means that the RMS input power must exceed the specification for 1 second before the amplifier will shut down.
The default value: THD=12000 (approximately three seconds). THD can not be assigned to a variable.

In addition, the SmartMotorTM servo motor also monitors its internal temperature. If the internal temperature exceeds a programmable set point, the amplifier shuts down and indicates an overheat error (see status bit Bh). The SmartMotorTM servo motor will remain in an overheat condition until the internal temperature drops 5° C below the programmable set point. This set point is determined by the function TH. The valid range for **`TH`** is 0 to 70, with units in degrees Celsius.

For example, if you enter, TH=50 the amplifier will indicate an overheat if the internal temperature reaches 50°C and will come out of the overheat condition when the temperature drops below 45°C.
The default value for **`TH=70`**.
THD can be stored into a variable. For example, t=TH will store the value of TH to the variable t.

SECTION 1.6d: Temperature , Voltage and Current Monitoring

TEMPERATURE MONITORING: The Temperature and RMS power of the SmartMotor™ servo motor can be monitored for diagnostic, preventative maintenance reasons. The real time temperature is read by the TEMP function and is given in units of degrees Celsius. TEMP is read by storing it to any variable. For example,

`t=TEMP`	this command ill assign the internal temperature to the variable "t".
`Rt`	this command will report variable "t". (Note: Any variable name can used .)

VOLTAGE MONITORING: The bus voltage is monitored by the user "J" analog input using the UJA function. User J pin is not physically accessible by the user. UJA provides the input bus voltage in tenths of volts. The accuracy of the reading is +0.1VDC.
For example, **`v=UJA`** will assign the input voltage to the variable "v".

If the reading is 336, the input voltage is 33.6+0.1 VDC.

CURRENT (I_{RMS}) MONITORING: The RMS current is monitored by the user "I" analog input port using the **`UIA`** command. User defined port I pin is not physically accessible by the user.
UIA will provide the measured RMS current in hundredths of ampere.
The accuracy of the reading is 0.1A.

For example, **`i=UIA`** will assign the RMS current to the variable i.
If the reading is 234, the measure current is 2.34+0.1 Amps.

SECTION 1.7: Servo-Amplifier OFF Command
On version 4.00 and newer motors you end the motor from servo-ing place by shutting off the servo-amplifer by issuing "OFF". This will turn off the servo-amplifer and the shaft will be able to rotate freely .

`OFF` Turn motor servo off.

SECTION 1.8: External Program Memory Chip
SmartMotors are available with internal memory on the "D" type I/O connector. On the SQUARE motors (Molex type connectors) the memory chip is external.

External program memory modules (EEPROM) are now available in 32k byte and 8k byte for square type motors.
D type motors (with 15 pin I/O connector) come standard with internal program memory.
Please refer to Figures 1.81 and 1.82 for Square and D-type motors.

Typical "SQUARE" Type Motor:

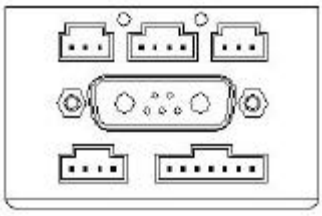


Figure 1.81: SQUARE Type Motor (Molex™ connectors and 7pin-combination connector)

Typical D-Type Motor:

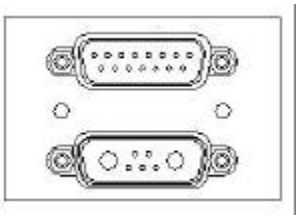


Figure 1.82: D-Sub Type Motor (15pin D-sub connector and 7pin-combination connector)

SECTION 2.0: ADDITIONAL MOTION MODES

SECTION 2.1a: Mode Follow with ratio, "electronic gearing"

SmartMotor™ users wanted variable mode follow ratios. To address that need with a fixed point system, we implemented a numerator/denominator function. This uses an 'integer fraction' to provide the equivalent of a floating point relationship between the motor and an incoming encoder, or step and direction, signal. For example, a mode follow relationship of 4.125 can be produced by setting MFDIV=8 and MFMUL=41.

Mode Follow with Ratio Commands

MFMUL=<expression>	Set numerator (signed 16 bit integer)
MFDIV=<expression>	Set denominator (signed 16 bit integer)
MFR	Engage Mode Follow Ratio

SECTION 2.1b: Mode Follow Phase Offset

A phase offset may be introduced when using the mode following commands. A phase offset allows the motor shaft to be aligned relative to the signal being followed. The offset is initiated by the G command, if the D command variable holds a (non-zero) offset. The V command variable controls the rate at which the offset is applied. The offset value in D decreases in magnitude until it reaches 0. V remains unchanged. The A command variable is used for internal calculations, so any prior value will be lost. If no offset is desired, be sure D=0 when issuing the G command while in MFR or MSR mode. While the offset is being applied, changes to D,V, or A will have immediate effect.

Mode Follow with Phase Offset Sequence:

MF4	Set counter mode (resets external counter to zero)
MFMUL=1	Set ratio multiplier
MFDIV=1	Set ratio divisor
MFR	Set Follow-Ratio Mode, initiating ratio calculation
D=0	Ensure there is no unintended phase offset
G	Start (applies new ratio to all subsequent external encoder changes)

Let's say we observe the motor shaft off ¼ revolution relative to the external encoder, both turning about 1 rps.

```
D=500     'Set positive ¼ revolution relative offset for motors with a 500 line encoder
V=410     'Want the relative offset to complete in 20 seconds.
          '(500 counts/20 seconds * 65536 scaled counts/count * 1 second/4000 servo samples =
          '410 scaled counts/servo sample).
G         'Begin phase offset
V=0       'Pause phase offset
RD        'Shaft move 299 counts relative to gear, 201 counts to go
V=410     'Resume phase offset
RD        'Reports Phase offset complete
```

SECTION 2.2: Mode Step and Direction and Mode Step with Ratio

MSR	Engage Mode Step Ratio
MFMUL=<expression>	Set numerator (signed 16 bit integer)
MFDIV=<expression>	Set denominator (signed 16 bit integer)
O=0	Set current shaft position as origin. (Uppercase letter O equal to zero)

SECTION 2.3: Mode Cam & Mode Cam with Ratio

For reciprocating motion such as the output from a variable profile CAM, there is a new mode initiated by the command "MC" (Mode Cam). This function allows a complex relationship between an incoming encoder signal and the motor's own encoder. It is described by as many as 100 sixteen bit words loaded in the array (aw[value]), and will linearly interpolate velocity & acceleration values between those points.
The BASE represents the number of external encoder counts to define one cam cycle or cam revolution.

BASE = <expression> *where: 2 <= BASE <= 32768*

The number of points in the cam profile is determined by the table entries given by:

SIZE =<expression> *where: 2<= SIZE <= 100*

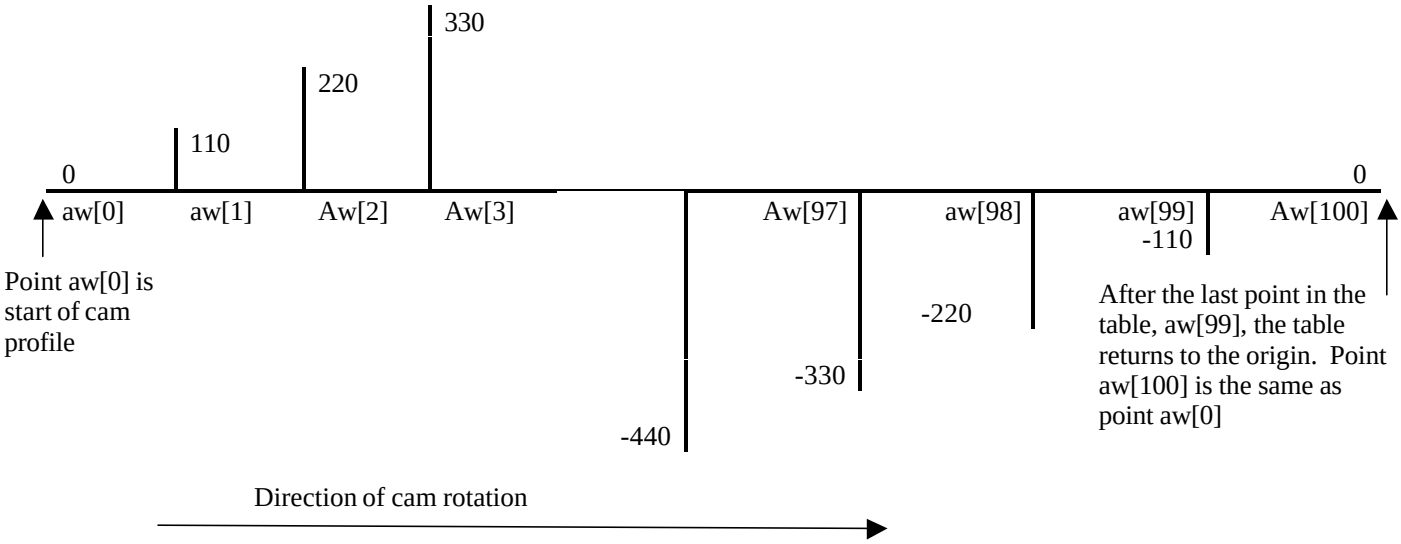
SIZE <= BASE

16 bit data values can be stored (values between – x to +x) to Cam Table.
They are to be stored in aw[0]…to aw[SIZE-1]. The data entries must represent a sample point at evenly spaced intervals of the required offset

Example Code:

```
MF4             'Reset external encoder mode and zero counter
BASE=4000       'Set the number of external encoder counts per cycle ( 4000 counts)
SIZE=100        'Tell how many array points, or table entries, to use
E=10000         'It may be necessary to increase error band to ensure the motor, when rotating at a high
                'speed, does not assumme it is seeing a position error as it tries to move to the next point
                'in the cam table.

                'CAUTION: ensure that the error band is not set so high that it allows the motor to
                'perform a violent move that damages machinery or personnel.

aw[0] 0 110 220 330 440 … -440 -330 -220 -110.   Load the array

                'The maximum line length is 128 characters. If more characters are required, use a
                'carriage return. At the last point in the table, aw[99] the 'cam' has almost completed
                'one revolution. At position aw[100] the cam has completed one revolution returns to the
                'first point in the table, the origin aw[0].. Note the array must end with a period.

MC              'Set Mode Cam, performs buffered calculation
G               'Start, resets motor position to zero – points at beginning of table and
                'engages CAM follow mode
```

The cam profile of this example is shown in the following sketch:



In this example, 10000 counts from the external encoder drives one 'rotation' of the cam. To continue cycling through the cam table, external encoder readings are translated to an offset within the range 0-10000. An external encoder measurement of 10350 translates to 10350 – 10000 = 350

Since the BASE = 10000 and there are 100 table entries, each table entry represents a segment of magnitude 10000/100 or 100 counts. The table entries, aw[ ], correspond to external encoder readings at 100 count intervals:

aw[0] 0, aw[1] 100, aw[2] 200, aw[3] 300, aw[4] 400,……..
……aw[96] 9600, aw[97] 9700, aw[98] 9800, aw[99] 9900

In our example, if the external encoder reading translates to 350, the sample point lies between aw[3] and aw[4]. The actual requested shaft position is calculated using linear interpolation.

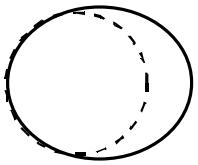
From the cam table:
aw[3] = 330
aw[4] = 440
If external encoder = 350,
Required position = sample position aw[3] + a fractional part of (aw[4] –aw[3])
Required position = 330 + ((350-300)/100 * (440 – 330) = 385
= 330 + 50/100 * 110
= 330 + 55 = 110

Note: The SmartMotor sets its position to ZERO (P=0) where the mode is engaged. The table offsets are relative to ZERO.
The external counter may reverse at any time.
Both positive and negative external encoder readings map to the data table.
The normal PID update rate (4 kHz) is maintained.
The required position calculation is rounded to the nearest encoder count.

EXTREME CAM PROFILES

At the end of each cam revolution, the table returns to the origin. Remember that abrupt changes in cam profile, or increases/decreases in cam diameter, will require high acceleration from the motor to ensure the shaft moves to the required angular position during one interval. Extreme changes could cause error problems, especially if the system is rotating at high speed.

Cam profile with smooth transitions



Cam profile with abrupt transitions

CAM MODE COMMANDS

Mode cam can multiply encoer inputs by issuing the follwowing commands.

MC2	Mode cam with final position request multiplied by 2
MC4	Mode cam with final position request multiplied by 4
MC8	Mode cam with final position request mutilplied by 8

SECTION 2.4: Coordinated Motion (Multi-axis Contouring Applications)

Coordinated Motion is a new feature of SmartMotors that have firmware versions 4.15 ,4.75, 4.40,4.41 and later. It allows SmartMotors on a daisy chain to be synchronized and precisely controlled by feeding the SmartMotors coordinate and time data. The Coordinated Motion command prompts for an ASCII formatted coordinate file (save as ***.txt*** file). The file must only have integer values separated by tabs:

	motor 1:	*motor 2:*	*motor 3:*
line 1:	**position clock**	**position clock**	**position clock**
line 1:	**position clock**	**position clock**	**position clock**

Sample file:

```
0     0     0     0
100   128   -100  128
200   256   -300  256
400   384   -100  384
800   512   0     512
```

The difference between position values must not exceed 8,388,607, otherwise an invalid position delta error will occur. The difference between clock values must not exceed 32,768 otherwise an invalid time delta error will occur. The clock values must also differ by values that are powers of 2 (for example 1, 2, 4, 8, 16, 32, 64, 128, .. , 32768). The Coordinated Motion command automatically addresses the SmartMotors on a daisy chain according to their position on the daisy chain.

If the first clock values are zeros, SMI assumes the position coordinates are relative to the current trajectory position (not the immediate position as reported by RP). Otherwise, the coordinates are absolute, and movement is done from the current trajectory position (not RP) and the first clock value is the "homing" time.
To return the trajectory position, issue the commands: aa=P, RP.

Note that the immediate position (returned by RP) and the trajectory position (variable P) must be the same, otherwise a position error may occur if the position differs by more than the error band E.

When the file is being sent and Coordinated Motion is in progress a dialog box pops up.
You can press the escape key or click anywhere in the dialog box to cancel Coordinated Motion.
You should not perform any CPU-intensive operations while running Coordinated Motion.
If SMI is not able to send data fast enough, possibly because another program interrupts it, a buffer underflow error will occur. If you cancel the motion or the file runs out of coordinates, the motors are sent an S command to halt motion and servo in place. When the trajectory clock values end for a motor, an S is sent to have the SmartMotor servo in place.

For more advanced functionality of Coordinated Motion, like dynamically generated coordinates, etc., the SMIEngine toolkit has been developed. SMIEngine is a component object that implements various primitives for communicating with SmartMotors and gives full control of Coordinated Motion.

SMI is supplied with an executable CMotion.exe that is used to control Coordinated Motion. The command-line format for CMotion.exe is: "CMotion portname filename" (ex. "CMotion COM1 c:\test.txt")
(Do not include quotation marks in the arguments.) This is the same executable that is run from the SMI Tools menu.

SECTION 3: INPUT/OUTPUT FUNCTIONS

In addition to SmartMotors special functions (right limit, left limit, step & direction ...etc) ports A to G are equipped to process Inputs/outputs.

SECTION 3.1: Software Selectable Functions

Each of the I/O pins have many devices attached internally that perform different I/O functions at each pin. I/O pins A to G can receive analog input (0-5V, 10 bit analog-to-digital converter), digital input (5V TTL) and send digital output (5V TTL). These functions are software selectable. Each I/O pin can change the selected function at anytime. To use a particular I/O function issue I/O commands below. Please refer to section 3.2 for sample code for I/O initialization.

INPUT /OUTPUT COMMANDS

New flexibility and functionality has been added to all I/O:
List of I/O commands

UA= expression	(set pin OUT LATCH to lsb, 0 or 1)
UAI	Assign pin A to input state
UAO	Assign pin A to output state
variable**=UAA**	Read pin a as 10 bit analog input
UB=expression	(set pin B output latch state)
UBI	Assign pin B to input state
UBO	Assign pin B to output state
variable**=UBA**	Read pin B as 10 bit analog input
UC= expression	(set pin C output latch state)
UCI	Re-assign right limit to input state
UCO	Re-assign right limit to output state
variable**=UCA**	Read pin C as 10 bit analog input
UCP	Restore pin to right (plus) limit function
UD=expression	(set pin D output latch state)
UDI	Re-assign left limit to input state
UD0	Re-assign left limit to output state
variable**=UDA**	Read pin D as 10 bit analog input
UDM	Restore pin D to left limit (minus limit) function
UE= expression	(set pin E output latch state)
UEI	Assign Anilink Data pin to input state
UEO	Assign Anilink Data pin to output state
variable**=UEA**	Read pin E as 10 bit analog input
UF= expression	(set pin F output latch state)
UFI	Assign Anilink Clock pin to input state
UFO	Assign Anilink Clock pin to output state
variable**=UFA**	Read pin F as 10 bit analog input

UG	Restore G sync line functionality
UG=expression	(set pin G output latch state)
UGI	Assign pin G to input state
UGO	Assign pin G to output state
variable**=UGA**	Read pin G as 10 bit analog input

SECTION 3.2: SAMPLE I/O PIN INITIALIZATION AND READ/WRITE SEQUENCE

Examples show: initialize Port B as an Analog Input, initialize Port C as an Analog Input and initialize Port D as an Digital Output

EXAMPLE 1) Digital Input: initializing Port A as an Digital Input .

Note: The SmartMotor Input ports have an internal 5kΩ pullup resistor, default state of input port is high and triggered when input line goes low. The exception is when **LIMH** limit High command issued triggers Left/Right Limit on high signal (version 4.15 & 4.40) .

```
UAI          'Initializes (U)ser defined port (A) as an (I)nput
b=UAI        ' Store digital input (at this instant in time) into variable "b"
Rb           ' Reports ( prints on screen) the value stored in the variable "b"
dd=UAI+5     ' Math operation is allowable on .
```

EXAMPLE 2) Digital Output: initializing Port B as an Digital Output .

```
UBO          'Initializes (U)ser defined port (B) as digital (O)utput
UB=1         'Sends logical high output to I/O pin B
             'Latches that logical state until new state is outputed.
UB=0         'Sends logical low output to I/O pin B.
```

EXAMPLE 3) Analog Input: initializing Port B as an Analog Input .

```
             'Initializes (U)ser defined port (E) as an (I)nput
c=UEA        ' stores analog input value at this instant in time
             ' into variable "c" (value of A/D Converter is 0 to 1023 ).
             'Sets (U)ser defined port (E) as (A)nalog input (10 bit A/D Converter).
             ' you can store the analog input into any of the variable names.
Rc           ' Reports (prints on screen) the value stored in the variable "c"
ee=UEA*100   'Math operation on the analog input is allowable
```

SECTION 3.3: Motor I/O Connector types:

SmartMotors have two types of I/O connectors:

D-SUB type (15pin D-sub connector and 7pin-combination connector)
SQUARE type (Molex™ connectors and 7pin-combination connector)

Refer to Figure 3.31 for typical D-sub Motor and Figure 3.31 for typical D-sub Motor.

Typical D-Type Motor:

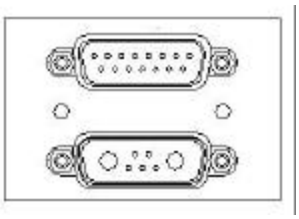


Figure 3.31: D-Sub Type Motor (15pin D-sub connector and 7pin-combination connector)

Typical "SQ" Type Motor:

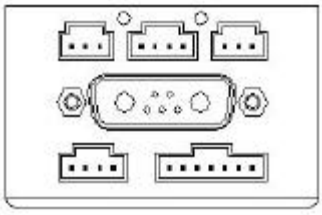
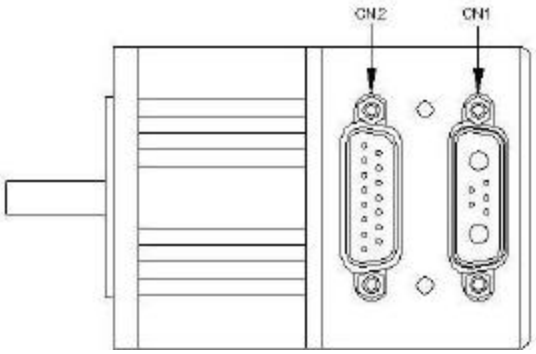


Figure 3.32: SQ Type Motor (Molex™ connectors and 7pin-combination connector)

SECTION 3.4: I/O Connector Pinouts

This section provides pinouts for the D-sub connector and Molex I/O connector. Connector 1 and 2 pinout is listed below.

D-sub Connector Pinouts



CN1: 7 PIN COMBO D-SUB MALE CONNECTOR		
PIN#	NAME	FUNCTIONS
1	Pin G	Go-Synchronization Pin
2	+5V OUT	+5V Out
3	RS232 TX	RS232-Transmit
4	RS232 RX	RS232-Receive
5	GND	Signal Ground
A1	POWER	Power,+24v to 48v
A2	PWR GND	Power Ground

CN2: 15 PIN D-SUB FEMALE CONNECTOR					
		SOFTWARE SELECTABLE I/O PIN FUNCTIONS			
PIN#	NAME			DIGITAL I/O (TTL)	ANALOG INPUT (10 bit)
1	Port A	Step Input	External Encoder A Input*	5V	0 to 5V
2	Port B	Direction Input	External Encoder B Input*	5V	0 to 5V
3	Port C	Right Limit Input		5V	0 to 5V
4	Port D	Left Limit Input		5V	0 to 5V
5	Port E	Anilink Data	RS-485 Signal A	5V	0 to 5V
6	Port F	Anilink Clock	RS-485 Signal B	5V	0 to 5V
7	Port G	Go-Synchronization Pin	RS-485 Adapter Control Line	5V	0 to 5V
8	ENCA OUT	Encoder A Output			
9	ENCB OUT	Encoder B Output			
10	RS232 TX	RS232-Transmit			
11	RS232 RX	RS232-Receive			
12	+5V OUT	+5V Out			
13	GND	Signal Ground			
14	PWR GND	Power Ground			
15	POWER	Power,+24v to 48v			

Figure 3.41: D-Sub Type Motor Pinout

(* SM2315D does not have external encoder capabilities)

Molex connectors SM17xx and 23xx (SmartMotor SQ)

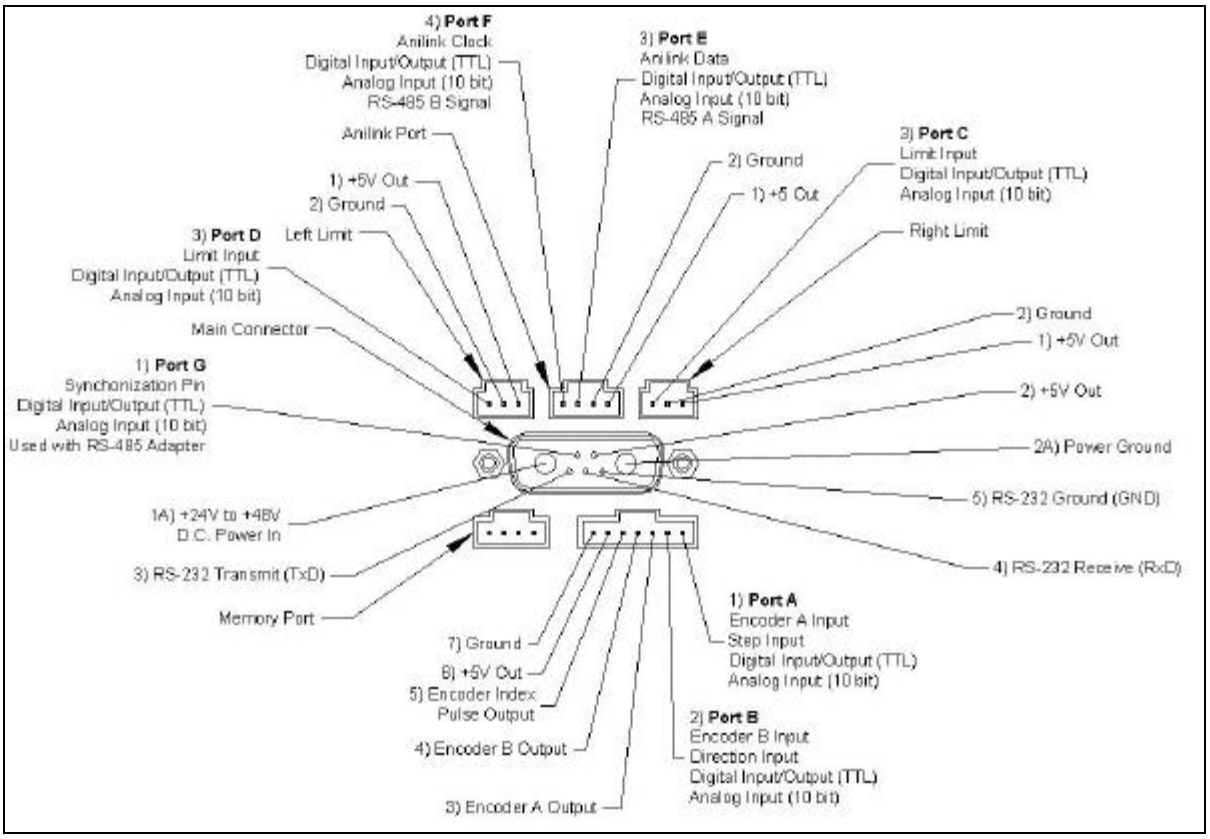


Figure 3.42: SQ Type Motor Pinout (SM17xx and SM23xx)

Molex connectors SM34xx (SmartMotor SQ)

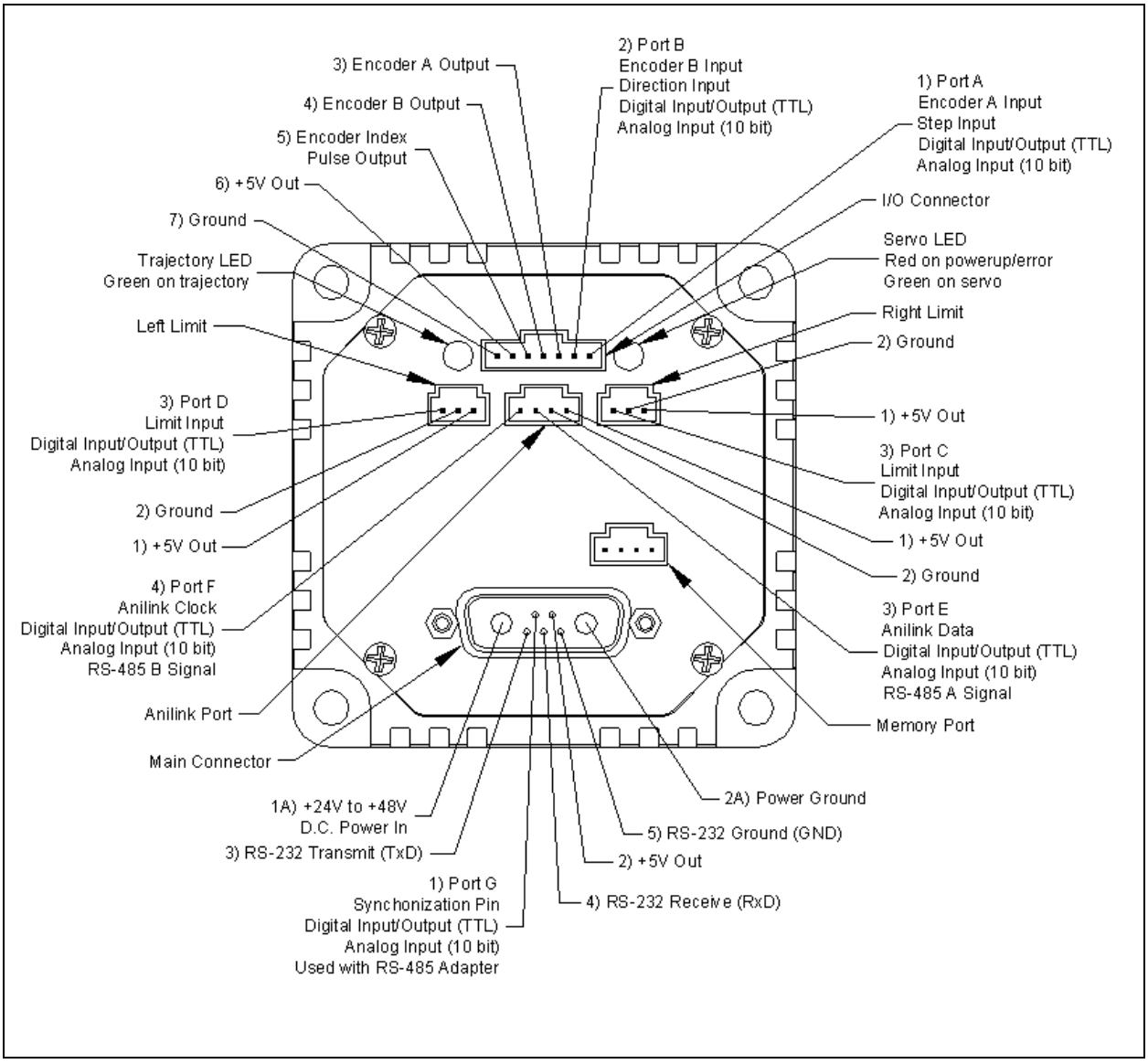


Figure 3.43: SQ Type Motor Pinout (SM34xx)

SECTION 3.5: D-sub motor Wiring diagram for Anilink peripherals

The D-sub type use port E and F as the Anilink port (Anilink clock line and Anilink data line). Figure 3.41 shows how to make a cable connecting D-sub motor to Animatics Anilink peripheral.

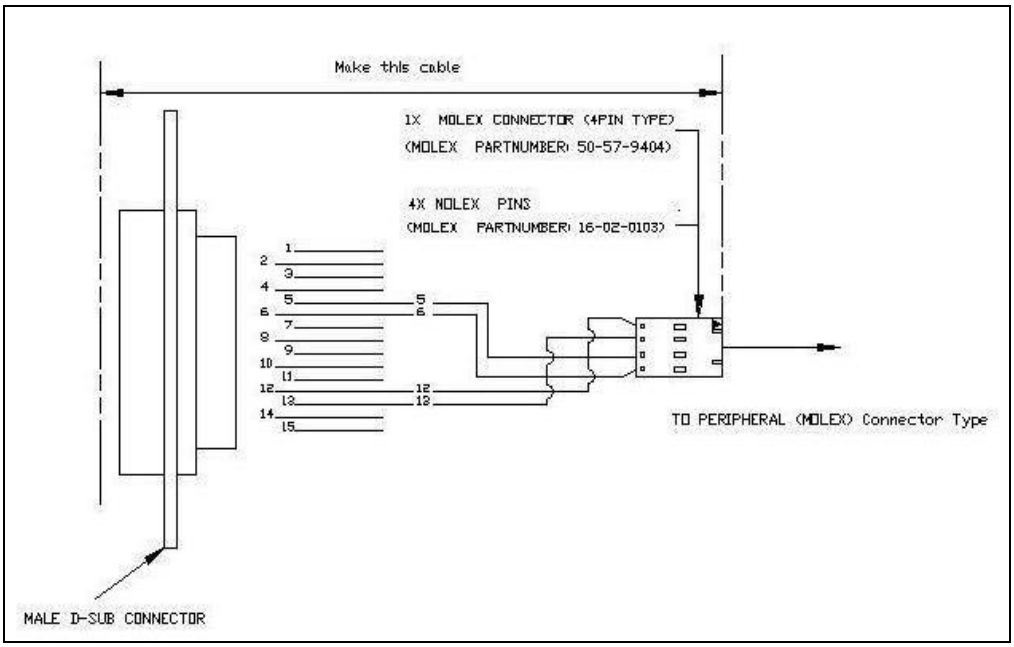


Figure 3.51: Dsub Cable To ANILINK 4 Pin Molex Connector

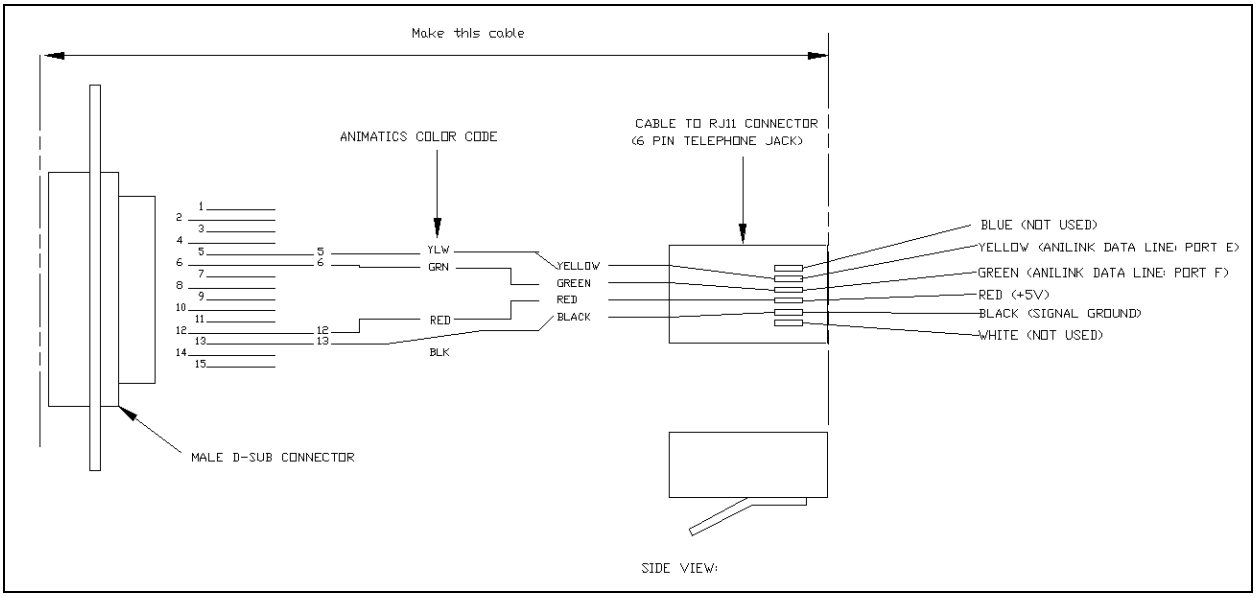


Figure 3.52: Dsub Cable To ANILINK RJ11 Connector

SECTION 4.0: PROGRAMMING LANGUAGE

SECTION 4.1a: Variable and Arrays Names

Variable Names

The number of variables have been expanded dramatically. The user now has the following 77 variable names at their disposal:

`a to z`	32 bit (first set of 26 variables)	(32 signed bits)
`aa,bb, cc ,dd…..to …zz`	32 bit (second set of 26 variables)	(32 signed bits)
`aaa, bbb, ccc to …yyy`	32 bit (third set of 25 variables)	(32 signed bits)

Note : Variable names with non-duplicated letters such as : ab, bc,cd,….abc, cde,efg,…etc… are not available.
These variable are 32 bit signed that means the can store values from - 2,147,483,647 to 2,147,483,647.

Arrays Names

As an alternative to the above two and three letter variables, sharing the same data space, the following arrays could be used. The same space can be used for three different variable types: 8 bit; 16 bit; & 32 bit. Naturally, the number of variables yielded from the space varies in accordance with the variable data lengths.

`ab[i]` Can store value of -128 to 127 (8 signed bits)
where i = 0...199
overlays aa-zz & aaa-zzz

`aw[i]` Can store values -32,768 to 32,767 (16 signed bits)
where i = 0...99
overlays aa-zz & aaa-zzz

`al[i]` Can store values from - 2,147,483,648 to 2,147,483,647 (32 signed bits)
where i = 0...49
overlays aa-zz & aaa-zzz

Where "i" can be a variable name a to z or a constant.

Reporting Variables

The array variables can be read from a host just as the traditional variables are, by prefixing the variable name with a capital "R". This is also known as the Report command which prints the value onto the communication line.

`R`*_variable_name*

For example: **`Rab[4]`** Reports Array Byte (**`ab[...]`**) 4th array location
`Rd` Report value stored in variable "d"

For flexibility, a long (signed 32 bit number) can be written and four bytes read from the same space, or visa-versa.

SECTION 4.1b: Initializing Variables and Arrays

Because of the vastly greater number of variables and additional variable types, we have implemented a convenient way of initializing a series of variables on a single line. It is shown in the following example. Note: the period after the last line entry the end of variable initialization on that line:

Example

```
a 1 -2 13 24 35.
```

Performs:

```
a=1
b=-2
c=13
d=24
e=35
```

```
aw[12] 15 20 30.
```

Performs:

```
aw[12]=15
aw[13]=20
aw[14]=30
```

SECTION 4.2: Long Term Variable Data Storage (VST,VLD Commands)

In addition to the program memory there is an additional internal 8k EEPROM chip for the long term storage of data (not standard on some models, please refer to product selection chart). It is accessed by first locating a pointer into that memory space and then doing single or bulk stores and loads. The associated commands are:

EPTR	Locates the EEPROM pointer, range (0……..7999)
VST	Stores data at that location and forward
VLD	Loads (retrieves) data from that location and forward

Example: Storing values to data memory (VST)

EPTR=1000	Set the pointer to 1000 to a memory location
VST(aw[15],5)	This command stores five words from the array starting with aw[15]. Each word is two bytes, the five words will take up10 bytes of space. Memory required for the variable type is automatically allocated.
p=15	In this example we initialize variable p and q
q=5	
EPTR=2000	Set the pointer arbitrarily to 2000
VST(ab[p],q)	You can use variables in place of constants

Example: Reading values from the data memory (VLD)

EPTR=1300	Set the pointer to read the required memory address in the EEPROM
VLD(t,6)	This command reads six variables from the EEPROM and writes to t and then u,v,w,x, and y.
VLD(aw[10],3)	Read three words from EEPROM and store to aw[10], aw[11], aw[12] in RAM.

SECTION 4.3: Control Flow

The following commands can be used with Version 4.00–Q3 and below series SmartMotors™
The number of subroutine labels have been expanded. Following are the new limitations:

GOSUBn	Where n = 0...999	(This results in 1000 possible subroutine labels)
GOTOn	Where n = 0...999	
Cn	Where n = 0...999	(Subroutine labels)
STACK	Reset user program stack pointer (Repairs stack damage from GOTOing out of a nested subroutine)	

The "IF" statement now has optional "ELSE" and "ELSEIF.

Example 1): Gosub Statements

```
IF a==1
      GOSUB1
ELSEIF a==2
      GOSUB2
ELSE
      GOSUB3
ENDIF
```

Example 2): Switch Statement

A "SWITCH" structure has been added. This switch statement does the same to the example 1. Do not use variable "zzz" if using switch statement.

```
SWITCH a                  'Depending on value of variable "a"
                          'the following cases
                          'select.
CASE 1                    ' If a=1.
        GOSUB1            'go to subroutine 1
        BREAK             'break out of switch statement
CASE 2                    'If a= 2.
        GOSUB2            'go to subroutine 2
        BREAK             'break out of switch statement
DEFAULT                   'Where "a" is anything else
        GOSUB3            'go to subroutine 3
        BREAK             'break out of switch statement
ENDS                      'END of Switch statement
```

There is no limit to the number of WHILE…LOOPs statement.

SECTION 4.4: System Status Bits

State Variables:
State variables are binary variables either set (1) or reset (0). They are reported using the **RB**<bit name> command. Note, some RB<?> are not supported by VRE High Resolution encoder.
For example: **RBe** 'reports the excessive error bit

Report Status Byte/Word Command:
RS Report status byte
RW Report status word

STATUS BYTE:

Bit Name		DESCRIPTON	BIT#	VALUE	COMMENT
Bo	=1	Motor is OFF	Bit7	128	Real time
Bh	=1	Excessive temperature	Bit6	64	Real time
Be	=1	Excessive position error occurred	Bit5	32	Reset by "G"
Bw	=1	Wraparound occurred	Bit4	16	reset by G,MT,&Zw
Bi	=1	Index report available	Bit3	8	reset by RI
Bl	=1	Historical left limit	Bit2	4	reset by Zl, ZS, RS & RW
Br	=1	Historical right limit	Bit1	2	reset by Zl, ZS, RS & RW
Bt	=1	Trajectory in progress	Bit1	1	real time

STATUS WORD:

Bit Name		DESCRIPTON	BIT#	VALUE	COMMENT
Bk	=1	Over current state occurred	Bit15	32768	real time
Ba	=1	Over current state occurred	Bit14	16384	reset by Za & ZS
Bs	=1	Syntax error occurred	Bit13	8192	reset by Zs & ZS
Bu	=1	User array index range error occurred	Bit12	4096	reset by Zu & ZS
Bd	=1	User math overflow occurred	Bit11	2048	reset by Zd & ZS
Bm	=1	Left limit asserted	Bit10	1024	real time
Bp	=1	Right limit asserted	Bit9	512	real time
Bx	=1	Hardware index input level	Bit8	256	real time
Bo	=1	Motor is OFF	Bit7	128	real time
Bh	=1	Excessive temperature	Bit6	64	real time
Be	=1	Excessive position error occurred	Bit5	32	reset by G
Bw	=1	Wraparound occurred	Bit4	16	reset by G,MT,&Zw
Bi	=1	Index report available	Bit3	8	reset by RI
Bl	=1	Historical left limit	Bit2	4	reset by Zl, ZS, RS & RW
Br	=1	Historical right limit	Bit1	2	reset by Zl, ZS, RS & RW
Bt	=1	Trajectory in progress	Bit0	1	real time

COMMUNICATION STATUS BITS:

Bit Name		DESCRIPTON
Bb	=1	Parity error occurred
Bc	=1	Communication overflow occurred
Bf	=1	Communications framing error occurred

States Resets: State variables are reset to zero with the following commands.

Za	Reset hardware current limit violation indication.
Zb	Reset serial data parity error indication.
Zc	Reset communications buffer overflow indication.
Zd	Reset user math overflow indication.
Zf	Reset communications framing error indication.
Zl	Reset left limit "seen" indication.
Zr	Reset right limit "seen" indication.
Zs	Reset user command syntax error indication.
Zu	Reset user array indexing out of range indication.
Zw	Reset wraparound indication.

Global Resets:

Z	Software reset; the user program counter, user variables, and all firmware states are reset.
ZS	Reset all individual user system bits listed above without performing a complete [Z] reset.

SECTION 4.5: Report Mode Commands

The following commands report (prints onto communication cable) function and mode of operation.

RF	Report last **F=<value>**
RCS1	Report and Clear 8 bit check sum of channel 1 communication bytes.
RMODE	Report SmartMotor MODE of operation the response will be one of the following:

P	Absolute position mode
R	Relative position mode
V	Velocity mode
T	Torque mode
F	Follow mode using MF1, MF2, or MF4
S	Step & direction mode
C	Cam mode
X	Follow mode or step & direction mode with ratio
E	Position Error exceeded error limit and motor shut itself off. Issue "G" to get out of error mode. Increase position error limit if necessary.
O	Letter "O" means Motor off

SECTION 4.6: Function Commands

The following commands can only be used with Version 4.11 and below series SmartMotors™

F=2	Disables error code characters "!h", "!1", and "!e" upon command error detection.
F=4	Report command responses are re-directed to RS485 channel 1(Port E and F) instead of to RS232 channel 0 (7 pin Main Connector).

SECTION 4.7: Program Execution Speed (CHANGING PID UPDATE RATE)

If necessary reducing the controller's PID update rate can increase program execution speed. The user can slow the PID update rate with the new PID commands.

The commands are:

PID1	Invokes the default rate (4069 servo samples per second on version 4.00 and up)
PID2	Divides the rate by two (2038 servo samples per second)
PID4	Divides the rate by four (1017 servo samples per second)
PID8	Divides the rate by eight (509 servo samples per second)

NOTES: Since velocity, acceleration and WAIT are functions of PID rate, modify these values to get equivalent motion or timing.

SECTION 4.8: Transmitting and Uploading User Programs on EEPROM

Commands are:

UP	Transmit stored SmartMotor program in tokenized format to the host terminal.
UPLOAD	Transmit program from motor's non-volatile memory to host terminal as seen in programing window.
ES400	Force EEPROMS to be read from & written to at 400 bits per second.
ES1000	Force EEPROMS to be read from & written to at 1000 bits per second. (Newer EEPROMS only)
RCKS	Return check sums. If an "**F**" appears following the check sums, there is a definite read/write error.

SECTION 4.9: LOCKP Command: Prevent Upload of User Program

LOCKP	Disables UP,UPLOAD commands (Host/Terminal Command Only). LOCKP can only be issued from a host computer.

SECTION 4.10: Program Checksum

User programs and subroutine jump tables both have stored checksums. The command RCKS emits the checksums followed by 'F' or 'P' to indicate Fail/Pass. System bit Bk reflects the EEPROM Write/Fail state. A new upload command, "UP", permits a byte for byte comparison to the compiled version of the user source code, since the RCKS pass response does not absolutely verify the stored EEPROM program. The original UPLOAD command recreates the original uncompiled user source code.

The commands are: **RCKS**

RCKS returns two checksums, from the label table and the program. The returned output is:

Label checksum Program checksum 'result' The result is either 'P' or 'F' for pass or fail.

RBk returns a system bit, k. If Bk = **1** it indicates the EEPROM is not verified. A return Bk =**0** indicates no EEPROM failure detected.

SECTION 5: RS232, RS485 and ANILINK COMMUNICATION CHANNELS

Section 5.1: CommunicationCommands

Dramatic changes have been implemented in the area of serial communications. The primary RS-232 channel has been enhanced so that it can input string data. By executing in the "DAT" command, the primary channel will no longer interpret commands, but rather input characters and place them in an input buffer for the user program to access. An RS-485 port has been added to the motor. It uses the same pins as the AniLink port. The AniLink port can be used for RS485 or the Animatics AniLink network. It is not possible to use both functions at the same time. The RS485 port can be opened with a vast array of different parameters and will input commands or data. It responds to the same commands with the addition of the channel number "1".

CMD	Define primary RS-232 port as command input port
CMD1	Define secondary RS-485 port as command input port
DAT	Define primary RS-232 port as data input only
DAT1	Define secondary RS-485 port as data input only
SLEEP1, WAKE1, TALK1, SILENT1	All for additional RS-485 Channel

To open a communication channel:

OCHN(COM TYPE,CHANNEL,PARITY,BAUDRATE,STOP BITS,DATA BITS,SPEC **)**

Example: **OCHN(RS2,0,N,9600,1,8,C)**	Primary host command channel 9600 7 pin combination D-sub connector.
Example: **OCHN(RS2,0,N,9600,1,8,D)**	Primary host data channel 9600
Example: **OCHN(RS4,1,N,38400,1,8,C)**	Second channel: port E (RS485 signal A) and port F (RS485 signal B).
Example: **OCHN(RS4,1,N,19200,1,8,D)**	Second channel.

To close a comm channel:

CCHN(TYPE,CHANNEL**)**

Example: **CCHN(RS2,0)**	close main channel, 7 pin combination D sub connector.
Example: **CCHN(RS4,1)**	close secondary channel, port E and port F.

To identify the existence of characters in comm channel buffer:

LEN, LEN1

To get characters from a comm channel:

GETCHR, GETCHR1

Example

```
IF LEN>0          'Ensure there are characters in the buffer before using GETCHR.
a=GETCHR          'Fetch a character from host communications channel buffer.

IF LEN1>0         'Ensure there a characters in the secondary channel (usually RS485 buffer) before using
                  'GETCHR1.
ab[3]=GETCHR1     'Fetch a character from secondary communications channel buffer to array
```

SECTION 5.2: Print to A Comm Channel

Because multiple comm channels exist now, enhancements were made in the PRINT commands:

PRINT(...)	print to the primary host channel (7 pin combination connector)
PRINTA(...) - PRINTH(...)	print to the AniLink device address A to H (address set by jumpers pin)
PRINT1(...)	print to channel 1 [Port E & F] (usually the default RS485 channel)

Examples:

1) Print to main connector [7 pin combination D-sub connector]

```
PRINT("Hello World")
```

2) Print to AniLink channel [port E (AniLink Data) and port F (AniLink Clock)]
Example: send print message to LCD address B, starting at beginning of second line.
Refer to data sheet for program code specific to your Anilink peripheral device.

```
PRINTB(#148,"Hello World")
```

3) Print RS485 communication using Port E (RS485 signal A) and port F (RS485 Signal B) channel

```
PRINT1 (" T=40 MT ")
```

SECTION 5.3: Reporting Channel Protocol

Report channel commands exist to track the status of the communication channels:

RCHN	report comm channel status - all
RCHN0	report comm host channel status
RCHN1	report comm channel 1 status

Where bit0 = overflow error
bit1 = framing error
bit2 = command scan error
bit3 = parity error

ANIMATICS®

THE SMART USER'S GUIDE

UG 0002 (rev.1/02)

Animatics The SmartMotor™ User's Guide.

Please let us know if you find any erorrs or omissions in this work so that we may improve it for future readers. Such notifications should be sent by e-mail with the words "User's Guide" in the subject line sent to: **techwriter@animatics.com**. Thank you in advance for your contribution.

Contact Us:

Animatics Corporation
3050 Tasman Drive
Santa Clara, CA 95054
USA
Tel: 1 (408) 748-8721
Fax: 1 (408) 748-8725
www.animatics.com

TABLE OF CONTENTS

Continued on following page

TABLE OF CONTENTS

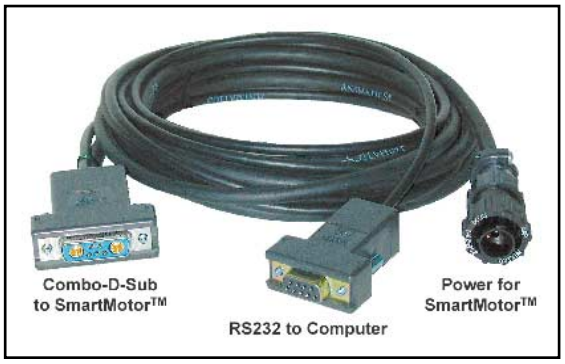
Continued from preceding page

A SmartMotor™ is delivered without any peripheral equipment due to the huge variety of systems and machines it can be used in. In order to make the SmartMotor run, the following will be needed at a minimum:

1. A SmartMotor™
2. A computer running MS Windows 95/98, 2000 or NT
3. A DC power supply for the SM1700 through SM3400 series motors or AC power cord for the SM4200 through SM5600 series motors.
4. A data cable to connect the SmartMotor to the computer's serial port
5. Host level software to communicate with the SmartMotor

The first time user of the SM1700 through SM3400 series motors should purchase the Animatics SMDEVPACK. It includes the CBLSM1-10 data and power cable, the SMI software, the manual and a connector kit.

The CBLSM1-10 cable (right) is also available separately.



Optional SmartMotor™ cable (CBLSM1-10)

Animatics also has the following DC power supplies available:

PS24V8A (24 Volt, 8 Amp) and PS42V6A (42 Volt, 6 Amp)

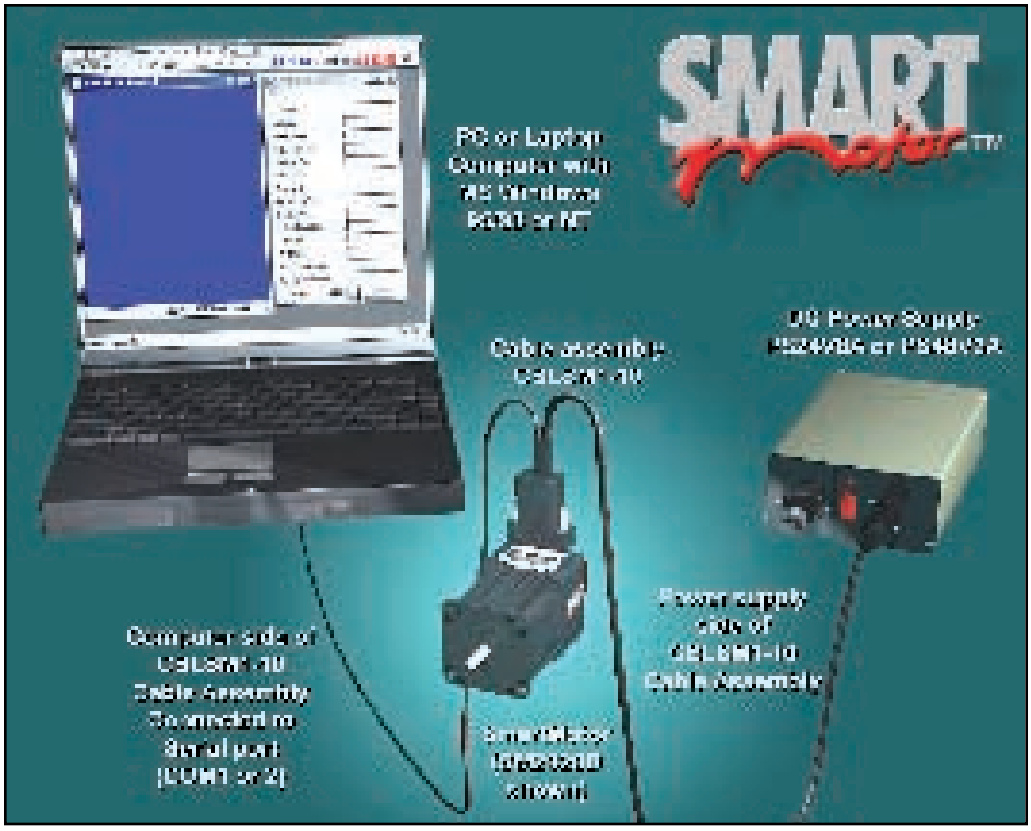
Optional PS24V8A or PS48V6A power supply

Both power supplies will work with the DC motors, but the PS42V6A supply allows the motors to operate at higher speeds. Note that the Speed-Torque curves are taken at 48VDC, the upper limit. Special care must be taken when near the upper limit or in vertical applications that can back-drive the SmartMotor. Gravity influenced applications can turn the SmartMotor into a generator and back-drive the power supply voltage above the safe limit. Many vertical applications require a SHUNT to protect the SmartMotor from damage. Larger open frame power supplies are also available and may be more suitable for cabinet mounting.

"Many vertical applications require a SHUNT to protect the SmartMotor from damage"

For the AC SmartMotors, SM4200 through SM5600 series, Animatics offers:

CBLSMA1-10	10' communication cable
CBLAC110-10	10' 110 volt AC single phase power cord



Connecting a SM2320D SmartMotor using a CBLSM1-10 cable assembly and PS24V8A power supply

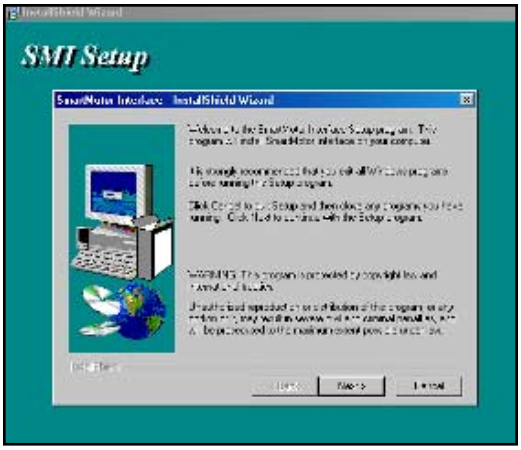
CBLAC200-10 10' 208-230 volt AC 3 phase power cord

SOFTWARE INSTALLATION

***Appendix F** is a complete screen by screen software installation guide*

Either the downloaded software (see **Appendix E**) or the SmartMotor CD-ROM software can be installed with the following procedure. Any differences are noted.

Double click the downloaded file, **SMISetup.exe**, or **Setup.exe** on the SmartMotor CD-ROM. The ***SMI Setup*** window (right) should now be visible.



*The **SMI Setup** window*

If the defaults are accepted on all of the setup screens after this screen, the software will be installed in the **C:/Programs/Animatics** directory. The **Programs** directory, that is accessed from the Windows **Start** button, will now have **Animatics** listed as an option with four subdirectories;

SmartMotor Interface

SmartMotor Tuning Utility

SMI Help

SMIEngine Help

A QUICK LOOK AT THE SMARTMOTOR INTERFACE

The **SmartMotor Interface** (**SMI**) is part of a suite of programs, that can be downloaded free from our web site (***www.smartmotor.com***) or purchased from Animatics Corporation on CD or a set of floppies. The suite runs on the **MS Windows 95/98, Windows 2000** or **NT** operating systems.

*The **SMI** suite is a free download from our web site (see **Appendix E**) or it can be purchased on a single CD or a set of floppies.*

The software connects a SmartMotor, or a series of SmartMotors, to a computer or workstation and gives a user the capability to control and monitor the status of the motors directly from the computer. Every SmartMotor has an ASCII interpreter built in. It is not necessary to use **SMI** to operate a SmartMotor. The interface does, however, allow the user the ability to write programs and download them into the SmartMotor's non-volatile EEPROM memory.

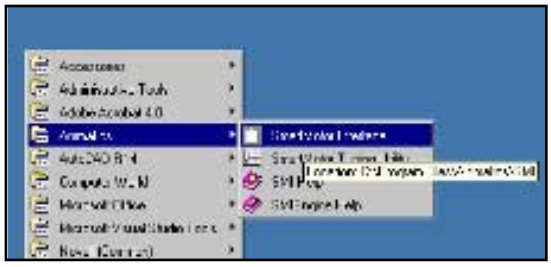
*Every SmartMotor has an ASCII interpreter built in. It is not necessary to use **SMI** to operate a SmartMotor.*

The suite includes the **SmartMotor™ Tuning Utility**, **SMI Help** and **SMIEngine Help** in addition to the **SmartMotor™ Interface** itself. With these tools the SmartMotor(s) can be tuned, addressed, monitored, tested and run from one computer. The SmartMotor can operate in a variety of different configurations. One or more motors can be completely controlled from a host computer, or by a Master SmartMotor programmed to control the others. Alternatively, each SmartMotor can have an independent program. The SmartMotor has three basic functions: **Motion Generation**, **Program Execution** and **Communications**. Each SmartMotor can do all three simultaneously, although the user should take care in implementing communications so as to avoid data collisions when programming multiple SmartMotors to send data over the same network.

PREPARING TO RUN THE SMARTMOTOR

Make sure everything is connected properly (see diagram on preceding page) then turn on the power supply and computer. A red LED should light on the motor indicating that it is powered and ready to go to work.

*Selecting **SmartMotor Interface** from the Windows' **Start** button*



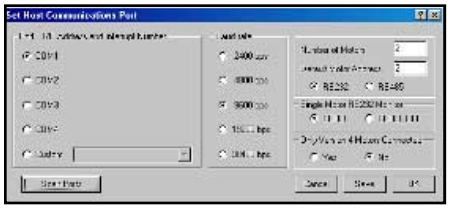
From the Windows desktop click on the **Start** button and then click on **Programs**, **Animatics** and the **SmartMotor Interface**.

***SmartMotor Terminal** window*

The dark blue **SmartMotor Terminal** window (right) should now be on screen. This is the main window that's used to communicate with a SmartMotor.

Click **Setup** on the menu bar, then click **Configure Host Port** from the drop-down menu. The **Set Host Communications Port** window should now be on screen.

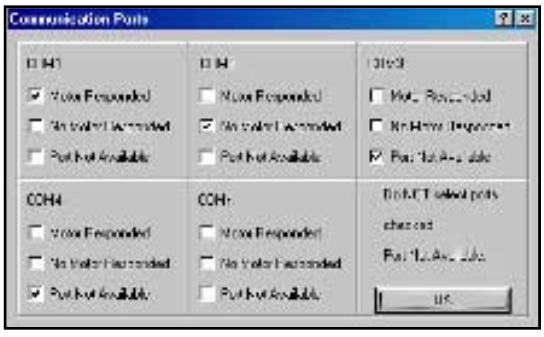
***Set Host Communications Port** window*



Select the **Scan Ports** button (lower left corner) and the **Communication Ports** window (below) should open.

This window only reports the status of the computer's serial ports and cannot be edited. If a SmartMotor is found (COM1 in the example) the **Motor Responded** box will have a check mark in it and the other two boxes under COM1 will be blank. If no motor is found (COM2) the **No Motor Responded** box will be checked. The computer used for these examples only has two COM ports (1 and 2) so the other COM ports (3 and 4) have **Port Not Available** checked. COMn isn't addressed.

***Communication Ports** window*



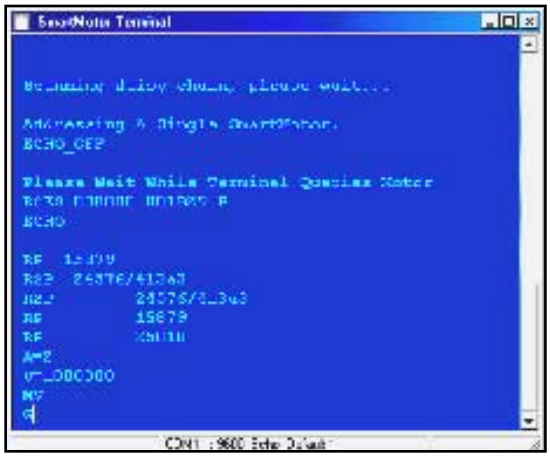
If the motor being tested returns similar results to those in the example, the computer and SmartMotor are properly interconnected and the SmartMotor is ready to transmit and receive data through its RS-232 serial port.

If no COM resource is available, independent measures should be taken to obtain an available port. This could be done either through system re-configuration or through the use of a USB to RS-232 or similar port converter.

MAKING THE SMARTMOTOR RUN

This is a quick tutorial designed to help the new **SmartMotor™** user get started. The commands used here are only described in enough detail to help in understanding what is happening.

Click on the button on the **SMI** tool bar. A few messages and responses will appear in the blue **SmartMotor Terminal** window (right).



The larger SmartMotors can shake and move suddenly and should be restrained for safety.

***SmartMotor Terminal** window with example data*

Now the computer and the SmartMotor are communicating and ready for commands. The example is showing the data returned from the SmartMotor used to get these screen shots. The data returned from a different motor may be quite different.

Transmitting commands

In the **SmartMotor Terminal** window type **RSP** (with CAPS LOCK *on*) followed by the enter key. The SmartMotor should respond with a string of data in the terminal window containing version information.

Type **RP** and then Enter. The motor responds with its current position (15879 in the example, above). Rotate the SmartMotor's shaft a little and type **RP** again. Note the change in the position (25010, above).

The **RP** (**R**eport **P**osition) and **RSP** (**R**eport **S**ample **P**eriod and version number) are Report to Host commands and are only two out of over sixty that can be used to query almost every aspect of a SmartMotor's status and performance. When any of the commands are entered into the **SmartMotor Terminal** window they return whatever data the command solicits.

Initiating motion

Now enter the following lines into the **Terminal** window (omitting the comments to the right).

```
A=100          'sets the Acceleration
V=1000000      'sets the maximum Velocity accelerated to
P=300000       'sets the target Absolute Position
G              'Go, initiates motor movement
```

After the final **G** command has been entered, the SmartMotor will accelerate up to speed, slew and then decelerate to a stop at the absolute target position.

1000000 Scaled Counts/Sample= about 1860 RPM for SM2300 series motors and about 930 RPM for series SM3400, 4200 and 5600 motors

Monitoring motor status

Click on the M button on the toolbar and the **Polling Motor 1** status window (right) will open. This window shows the current status of the test SmartMotor (number 1).

Polling Motor 1 dialog box. (The number refers to the motor number)

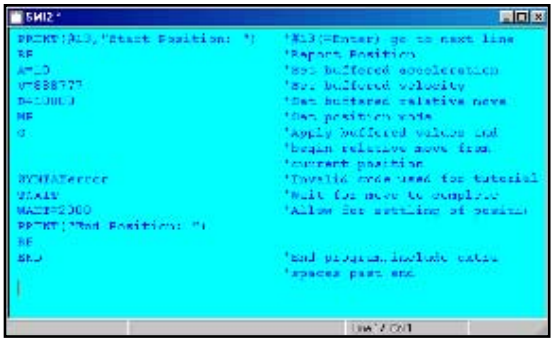
While the monitor is polling the motor, enter a new position and repeat the **G** command to initiate the move. With the monitor on, the progress of the move can be seen. The changing of the **Index Position** can also be seen as the encoder's index passes with each revolution (Motors equipped with a variable resolution encoder do not have an index marker).

To stop the motor in the middle of a move, enter the **X** command in the **SmartMotor Terminal** window. The SmartMotor will decelerate to a stop at the same rate the **A**cceleration parameter was last set (**A=100**).

Writing a user program

Press the button on the toolbar and the **SMI2*** program editing window (below) will open. This window is where SmartMotor programs are entered and edited.

Enter the following program in the editing window. It's only necessary to enter the boldface text. The text preceded by a single quote is a comment and is for information only. Comments and other text to the right of the single quotation mark do not get sent to the motor. The text is usually used to describe what's going on at that moment in the program. Pay close attention to spaces and capitalization. The code is case sensitive and a space is a programming element.



Program editing window.

```
PRINT(#13,"Start Position: ")   '#13(=Enter) go to next line
RP                              'Report Position
A=10                            'Set buffered acceleration
V=888777                        'Set buffered velocity
D=10000                         'Set buffered relative move
MP                              'Set position mode
G                               'Apply buffered values and
                                'begin relative move from
                                'current position
SYNTAXerror                     'Invalid code used for tutorial
TWAIT                           'Wait for move to finish
WAIT=2000                       'Allow for position settling
PRINT(#13,"End Position: ")
RP
END                             'End program,include extra
                                'spaces past end
```

*If the **SmartMotor** used in this test needs a memory module make sure one is installed.*

```
PRINT("End Position: ")
RP                  'Report Position
END                 'End program, include extra
                       spaces after END for test
```

After the program has been entered, select **File** from the menu bar and **Save as . . .** from the drop down menu. In the **Save File As** window give the new program a name such as "MyProgram.sms" and click on the **Save** button. The file name will replace the edit window title (**SMI2***).

Transmitting the program to a SmartMotor

To check the program and transmit it to the SmartMotor, click on the T button located on the tool bar. A small window will open with **Errors Found**. Click **OK** to close the new window. The error, "SYNTAXerror", should now be colored red. Click the > button and the cursor will jump to the line with the error.

Program editing window with "SYNTAXerror" highlighted

If there were additional errors the > button would send the cursor to the next error each time it was clicked. The < button steps through the errors backwards, the ¯ sends the cursor to the beginning of the program and the _ sends it to the end.

The bottom status bar of the editing window should now have three entries; **Found Next Error**, **UNRECOGNIZED TERM** and **Line 10 Col 1** (the present cursor position if it's located at the beginning of the line with the error).

To correct the error, use the cursor to highlight just the "SYNTAXerror" line, including the single quote and the comment, "Invalid code for tutorial" and press the delete key.

If the error had been a typo (**TWAI**P possibly) it could have been fixed by highlighting just the error (the P) and typing the proper letter (a capital **T**).

After the error has been corrected, click on the T button again. Some messages should start scrolling down the **SmartMotor Terminal** window and a progress bar showing **SMI** is transmitting the program to the SmartMotor should also be on screen (right).

Transmitting program to SmartMotor

Running the downloaded program

The programmed motor now returns data (the **PRINT** commands), and these characters can cause errors in the data returned to the **Polling Motor 1** window (page 10). To keep this from happening click on the **STOP** button at the bottom of the **Polling Motor 1** window and then click R on the toolbar to run the program for the first time.

The SmartMotor accelerates at the rate set (**A=10**) to the programmed speed or **V**elocity (**V=888777**) and then maintains that speed for the **D**istance (**D=10000**), determined by the number of motor shaft rotations, and then decelerates to a stop at the rate set by the **A**cceleration command (**A=10**). As the motor spins up watch the data in the **SmartMotor Terminal** window. Run the program several times by clicking on R and watch the results.

Because servomotors operate based on position error feedback, position may oscillate slightly.

Turn off the power to the SmartMotor and after a few seconds turn it back on. The motor will run the resident program every time it is turned on until a new program is downloaded. The SmartMotor will do this whether there's a computer connected to it or not. All it needs now is power.

TALKING TO SMARTMOTORS

The dark blue **SmartMotor Terminal** window opens whenever the **SMI**software is turned on. If the window has been closed, another operation requiring its use will open a new window.

To transmit a command to the SmartMotor in the **SmartMotor Terminal** window type the command on a new line and Enter on the same line.

Any line, or part of a line, can be transmitted through the serial port by moving the cursor to the line's end or anywhere along the line and pressing the Enter key. Everything to the left of the cursor position will immediately be transmitted to the SmartMotor and the cursor will jump to the end of the listing, past the last line of text and after a copy of what was just transmitted. If the highlighted line of text contains a macro, the macro will be expanded and transmitted as well. Macros are explained later in this chapter.

The Tab key sends the cursor to a new line after the last line of text without transmitting any data to the SmartMotor, no matter where the cursor was located in the **SmartMotor Terminal** window.

SMI is continuously monitoring the serial ports for data, so the asynchronous input is displayed by the **SmartMotor Terminal** window.

More than one command can be entered on a line when separated by a space. For example; **RA RV RP** reports **A**cceleration, **V**elocity, and **P**osition.

The SmartMotor Terminal status bar displays the COM port, baud rate, and Default Motor Address.

ECHO and ECHO_OFF Modes

The **SmartMotor Terminal** runs in either **ECHO** or **ECHO_OFF** mode. In **ECHO** mode, the SmartMotors echo back every character they receive and the **SmartMotor Terminal** refrains from displaying both the sent and received lines. When in **ECHO** mode, **SMI** will warn the user if it fails to receive the expected echoed response. In the **ECHO_OFF** mode, the terminal has no expectation of receiving an echo of the command sent. The **Send ECHO** and **Send ECHO_OFF** sub-items of the **Communicate** drop-down menu can serve to synchronize the **SmartMotor Terminal** and SmartMotor(s). Otherwise, if **SMI** is expecting an echo and the motor is not in echo mode, warnings will appear.

*The **SMI** software uses the ECHO feature to Auto-Address a Daisy Chain of SmartMotors.*

A single RS-232 SmartMotor can operate in either state, but a daisy chain of RS-232 SmartMotors must operate in **ECHO** mode for data to get through the network. RS-485 SmartMotors must operate in the **ECHO_OFF** mode or there will be data collisions.

Motors using RS-485 must address themselves by way of a stored program.

The third item on the **SmartMotor Terminal** status bar displays "Echo" if the Terminal is in the **ECHO** mode.

***Note**: When operating in ECHO_OFF mode the SmartMotor Terminal window has no way to synchronize with SmartMotors. If the values appear to be incorrect in the Polling Motor window, STOP the polling and START it again using the buttons at the bottom of the window.*

Addressing SmartMotors

In general, a single SmartMotor doesn't have to be addressed before using **SMI**. All of the motors in multiple SmartMotor installations (daisy chain) must be addressed first, otherwise the **SmartMotor Terminal** window may not work properly. The software stores information about each of the SmartMotors as their addresses are assigned.

Addressing the SmartMotor(s) can be done by selecting:

Communicate from the ***Menu bar*** and **Address Motor Chain** from the drop-down menu.

OR Click the button on the ***Toolbar***

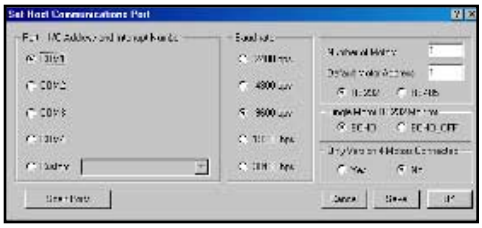
OR Enter an **&** (ampersand) character in the **SmartMotor Terminal** window.

Data returned after addressing a single SmartMotor

When a single motor is addressed it should return results similar to those shown in this window to the right.

Changing the baud rate

When switching to a different baud rate, change the motors baud rate first, and then change the **SMI** baud rate. The supported rates are 2400, 4800, 9600, 19200 and 38400.



Set Host Communications Port *window*

To change the SmartMotor's and **SMI's** baud rate, click on the **Communicate** drop-down menu and click on **Send New Baud Rate**. If the changes are to be used whenever **SMI** is started, click **Setup** on the menu bar and from that drop-down menu select **Configure Host Port** and then **Save** at the bottom of the **Set Host Communications Port** window. When the **Send New Baud Rate** command is chosen, the **Saving Baud Rate** window (left) gives instructions for saving the new settings.

If a SmartMotor is powered on with no program in its memory, it defaults to 9600 baud.

Saving Baud Rate *window*

Using Macros

Macros are used to store frequently used single commands or multiple command strings.

The maximum number of macros that can be recalled is 50, the maximum number of characters in a macro's name is 20 and the command string character limit is 220. Up to 4 Macros can be nested (a macro contained within another macro). The macro list is loaded when **SMI** is opened and saved when **SMI** is closed.

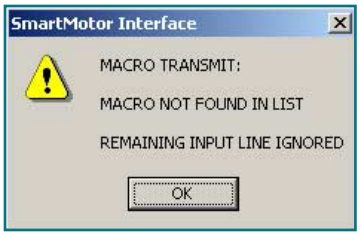
To define a macro use the following syntax followed by the enter key:

%MacroName Commands%

For example this line: %CmdS **RA RV RP**% defines a macro named **CmdS** which issues the **RA**, **RV** and **RP** commands (Notice the lower case letters in the title, **CmdS**).

To run a macro use the following syntax followed by the enter key: /MacroName

Macro's are case sensitive and the title must be entered exactly as it was typed. On the right is the error message received if a mistake is made.



Macro transmit error window. Click on **OK,** *retype the macro's name and try again*

To run the macro defined above enter the title exactly as it was typed: **/CmdS**

To list all of the defined macros select:

Communicate from the ***Menu bar*** and **List Macro Definitions** from the drop-down menu

OR enter **//** in the **SmartMotor Terminal** window.

To delete an existing macro type: -MacroName

To edit an existing macro, type: +MacroName

MONITORING SMARTMOTOR PARAMETERS

The **Polling Motor** window displays the SmartMotor's current status and parameter values.

To open the **Polling Motor** window, select:

Communicate from the ***Menu bar*** and **Monitor Status** from the drop-down menu

OR click on the M button on the ***Toolbar***

Polling Motor *window*

The most commonly used SmartMotor parameters are shown in this window. When the window is first opened, it immediately starts polling the SmartMotor(s) and updates the parameters.

Polling can be stopped with the **STOP** button and restarted with the **START** button at the bottom of the **Polling Motor** window. The **CLOSE** button closes the window. This window is read-only and cannot be edited.

Typing data into the **SmartMotor Terminal** window will pause the data stream to the **Polling Motor** window. Polling remains paused until the command is finally transmitted to the motor with the enter key.

If a program is written that transmits data through the serial port (the **PRINT** commands), it may conflict with the data displayed in the **Polling Motor** window. If this type of data is being used, stop polling.

Advanced Polling

The **Advanced Polling** window displays up to eight user defined parameters.

To open the **Advanced Polling** window select:

Communicate from the ***Menu bar*** and **Advanced Status** from the drop-down menu

OR click on the P button on the ***Toolbar***

Advanced Polling *window*

The value of a variable, status of a port or other customized parameters need to be monitored. Like the **Polling Motor** window, the **Advanced Polling** window (right) sends out report commands to the motor(s) in the background and shows the received responses in the window. If any of the blocks on the left side of the window are clicked, the **Standard Polling Variables** window will open.

These polling variables can be selected to replace the block clicked. For example click on the first block (the **a** block) in the **Advanced Polling** window, and click on the **KP** button in the **Standard Polling Variable** window. Then click on **CLOSE**. A window will open with instructions on how to save the advanced polling settings. Close the window to return to

the **Advanced Polling** window. Now the **a** in the first block is changed to **KP**. **P**ress the **START** button to start polling the motors again. The value of the **KP** parameter should be in the data box on the right side (250 in the example, right).

*Monitoring the **KP** parameter in the **Advanced Polling** window*

Note: Pressing any of the blocks on the left side of the Advanced Polling window will automatically stop polling. It can only be restarted with the START button.

At times it's necessary to monitor the status of a hardware port or an internal SmartMotor variable. Polling parameters can be customized using the **Advanced Watch Settings** window (below). To open the window, click on the **Advanced** button on the lower right corner of the **Standard Polling Variables** window.

The following is a description of each data entry box in the **Advanced Watch Settings** window:

***Advanced Watch Settings** window*

Command String: The actual data transmitted to the SmartMotor(s).

Caption String: Data displayed in the boxes on the left side of the **Advanced Polling** window.

Target Address: If there is more than one SmartMotor, enter the address of the motor to monitor.

Logical Mask: Enter the logical mask (in base 10) to apply to the SmartMotor response. If this value is "0" no mask is applied.

True caption: The data displayed if the final response value is not zero. The final value of the response is calculated by ANDing the SmartMotor's response with the logical mask value.

False caption: The data displayed if the final response value is zero. The final value of the response is calculated by ANDing the SmartMotor's response with the logical mask value.

Global Command Delimiter: The command delimiter in the command string. Usually commands are separated by one space.

Fetch Entry: Fetches current user defined configuration and updates the window entries.

Clear Entry: Resets the current entry to null settings.

Set Entry: Places the settings in current watch configuration

Close: Saves any changes and closes the window.

Advanced settings for monitoring custom parameters.

Advanced Polling Example

The following example shows how to set the advanced polling parameters to monitor motor data.

1. Click on the **a** box in the **Advanced Polling** window. The **Standard Polling Variables** window will open.
2. Click on the **Advanced** button. The **Advanced Watch Settings** window (right) will open.
3. Enter the commands to be sent to the SmartMotor in the **Command String** box. In this example the analog value of port C needs to be monitored. Note that there must be a space between **a=UCA** and **Ra**.
4. Enter a title for the string in the **Caption String** box (**Port C analog** in the example)

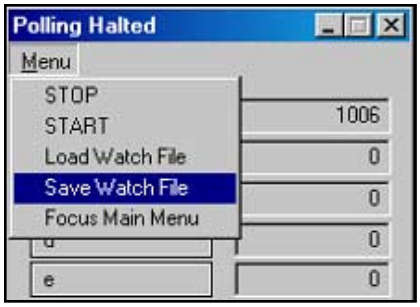
Monitoring the analog value of port C

5. If the SmartMotor is part of a daisy chain, enter the motor's address in the **Target Address** box and modify the title in the **Caption String** to reflect the motor being polled. If there is only one motor connected to the computer the **Target Address** box can be ignored.
6. Click on the **Set Entry** button, and then close the window. Close the **Standard Polling Variables** window.
7. In the **Advanced Polling** window, press the **START** button and watch the selected parameter polling (above).

***Note**: Lengthy command strings can cause communication errors. Advanced polling parameters are polled in the background and programs that send characters through the serial port during polling may cause problems.*

Saving and loading advanced settings

Saving and loading advanced settings



Advanced Polling window settings can be saved for future use. Click on **Menu** and select **Save Watch File** from the drop-down menu. In the save window, enter a name for the file. To load a previously saved file, select **Load Watch File** from the drop-down menu.

PROGRAMMING WITH SMI

This section focuses on the **SMI tools** used to create and edit SmartMotor programs. Later sections will focus on actual programming.

To create a new program file, select:

File from the ***Menu bar*** and **New** from the drop-down menu

OR Click on the button on the ***Toolbar***

OR Press the Ctrl+N keys.

A new window will open (below).

Program edit window

This is a simple text-editing window that's used to edit or create new programs. It works like all Windows based text editors with all of the Cut, Copy, Paste and Delete capabilities. When a new edit window is opened, it will have a default name (**SMI2** in this example).

*There are several sample programs in the **SMI Help** file. Just copy and paste them into the **SMI** text editor.*

Saving program files

The currently active program can be saved by selecting:

File from the ***Menu bar*** and **Save** or **Save As** from the drop-down menu.

OR Click on the button on the ***Toolbar***

OR Press the Ctrl+S keys for **Save** or Shift+Ctrl+S for **Save As**.

When saving a document for the first time, **SMI** displays the **Save As** window so the document can be named before it is saved. To change the name of an existing document, use the **Save As** command. There are two types of documents that can be saved by the **SMI** software:

- Program source files (.sms).
- ASCII source files (.src).

A **SmartMotor Terminal** window can be saved with file extension .mon, but it will be read only when it is reopened, which means it can't be modified and commands can't be sent to the motors through it. Only one **SmartMotor Terminal** window can be open at a time. To use a saved window, close the open **SmartMotor Terminal** window and open the saved window by clicking on **File** on the menu bar and then **Open** from the pull down menu and then double click on the file desired.

Loading program files

To open a previously saved program, select:

File from the ***Menu bar*** and **Open** from the drop-down menu.

OR Click on the button on the ***Toolbar***

OR Press the Ctrl+O keys.

Use this command to open an existing program source document or terminal document in a new window.

Use the **Open** command to open ASCII source files (with extension .src) if the **ASCII source** type was selected. The file is automatically renamed with a .sms extension. This replaces the **Import ASCII Source** command in older versions of **SMI**.

Building (Compiling) an SMI program

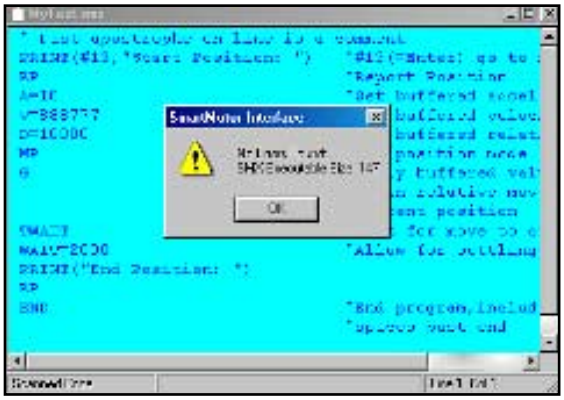
Compiling a program with no errors

Check and Compile a program by selecting:

Edit from the ***Menu bar*** and **Make SmartMotor Executable** from the drop-down menu.

OR Click on the button on the ***Toolbar***

OR Press the Ctrl+E keys.



The program is scanned for syntax and statement label errors. A window will show the end of scanning. If no errors were found, the first pane of the **Edit** window ***Status bar*** will have the **Scanned Code** message as **SMI** builds both the un-compiled .smx, and compiled, .sm, program files.

After the program has been compiled a small message box will be displayed stating: **No Errors Found: SMX Executable Size** followed by a number indicat-

ing the number of program executable bytes that will be used when this file is downloaded. Just click on **OK** to continue.

If errors are found, the message box will state **Errors Found** and the errors will be colored red.

Locating the Errors

The following are shortcuts for moving the cursor around the editing window:

To go to the beginning of the program, select:

Edit from the ***Menu bar*** and **Go To Top Of File** from the drop-down menu

OR Click on the button on the ***Toolbar***

OR Press Ctrl+Home on the keyboard

To go to the end of the program select:

Edit from the ***Menu bar*** and **Go To End Of File** sub-item from the drop-down menu

OR Click on the button on the ***Toolbar***

OR Press Ctrl+End on the keyboard

To go to the next error position select:

Edit from the ***Menu bar*** and **Find Next Error** from the drop-down menu

OR Click on the button on the ***Toolbar***

To go to the previous error position select:

Edit from the ***Menu bar*** and **Find Previous Error** from the drop-down menu

OR Click on the button on the ***Toolbar***

Downloading programs to a SmartMotor

To download a program to the SmartMotor select:

File from the ***Menu bar*** and **Transmit Program to Motor** from the drop-down menu.

OR Click on the button on the ***Toolbar***

OR Press the Ctrl+T keys.

If you press the button without Building or Compiling your program first, it will compile automatically, saving you the extra step.

Use this command to transmit the compiled source code of the presently active document window to the default-addressed SmartMotor.

This command first scans the currently active source document code for syntax and statement label errors. If any errors are found, they are treated as they were in the **Building (Compiling) an SMI program** section.

SmartMotor(s) must be addressed before a program can be downloaded to them.

If scanning the program finds no errors, the file will be compiled and transmitted to the addressed SmartMotor.

The integrity of downloaded code is validated by checksum. If the configuration defines more than one motor, then the user defined default motor will receive the file. Note that the source code comments are not sent to the SmartMotor.

If **SMI** finds an older SmartMotor not capable of handling a compiled file, it will issue a warning and transmit a .sm format file instead.

Note: *Older SmartMotors can be found with less or slower memory than has been shipping since roughly year 2000-on. Programs exceeding 8k should not be downloaded to the older 8k EEPROMS. SMI does not have the capability to determine the EEPROM capacity of a specific SmartMotor. Repeated checksum errors can be solved by issuing the ES400 command, slowing the programming and reading rate from 1,000bps to 400bps.*

Transmit Setup

*The **Transmit Setup** dialog box*

The **Transmit Setup** window (right) can be used to define the functionality of the **Transmit Program** command.

The **Transmit Setup** window is accessed by selecting:

> **File** from the ***Menu bar*** and **Transmit Setup…** from the drop-down menu.

It can be setup to always transmit the current program or a program stored in a file. When transmitting the current program, highlight the editor window that contains the program to send.

For a one-time occasion, use the **Transmit Now** button to use the currently displayed configuration and then press **Cancel** so the changes to the setup are not saved.

Check the **Transmit Version 3 Format** to transmit a current program in the **.sm** format.

Uploading a program from a SmartMotor

To upload a SmartMotor's program, select:

> **File** from the ***Menu bar*** and **Receive Program from Motor** from the drop-down menu

A new edit window will open and the SmartMotor's program will be uploaded to it. The program can then edited, saved and transmitted back to the motor. Note that the uploaded program has no comments. They were stripped off when the program was transmitted to the SmartMotor. Additional and unprintable compiler codes are removed during upload.

SMI MENU COMMANDS

This section describes the **SMI** ***Menu bar*** pull down menus. Many of these commands are also discussed in the **SMI** programming section.

File pull down menu:

File *menu*

New: (Ctrl+N) Create a new file in a new document window.

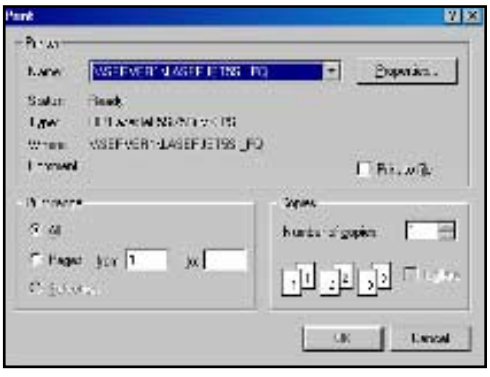
The file format is RTF (Rich Text Format). It cannot be read directly by an ASCII editor.

Open: (Ctrl+O) Open an existing document or terminal document in a new window. This command can be used to open ASCII source files (with extension .src) if the "ASCII source" choice was picked.

Open *File dialog*

Close: (Ctrl+W) Use to close an open document window. If a modified document is not saved, **SMI** displays a message asking whether the changes should be saved.

Save: (Ctrl+S) Use this command to save the active document to its current name and directory. When a document is saved for the first time, **SMI** displays the **Save As** window so it can be renamed.



Print *window*

Save As: (Shft+Ctrl+S) Use this command to save and name the active document. **SMI** displays the **Save File As** dialog where the document can be named and then saved.

Print: (Ctrl+P) Opens the **Print** window (right) where the number of copies to be printed can be set and the destination printer, and other printer setup options can be selected or changed. When **OK** is clicked the document will print.

Print Setup *window*

Print Preview: Disabled.

Print Setup: Opens the **Print Setup** window (left) where the printer, printer connection, paper size, and paper orientation can be selected and changed.

Transmit Program To Motor: (Ctrl+T) Download the program in the currently active window to the SmartMotor(s).

Transmit Setup: Adjust settings for downloading programs.

Receive Program from Motor: Receive a program from the default addressed motor to a newly opened window.

Exit: (Ctrl+Q) End the **SMI** application session.

Edit pull-down menu

__Edit__ pull down menu

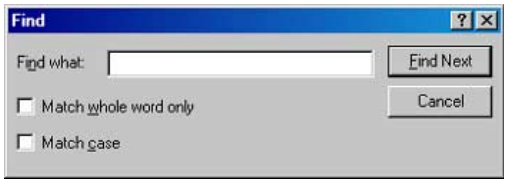
Undo: (Ctrl+Z) Reverses previous editing operation in the currently active document window, if possible. **SMI** provides only the most minimal support for the Undo operation.

Cut: (Ctrl+X) Deletes the highlighted data from the currently active document and moves it to the Windows clipboard.

Copy: (Ctrl+C) Copies the highlighted data from the currently active document to the Windows clipboard.

Paste: (Ctrl+V) Pastes data from the Windows clipboard into the currently active document.

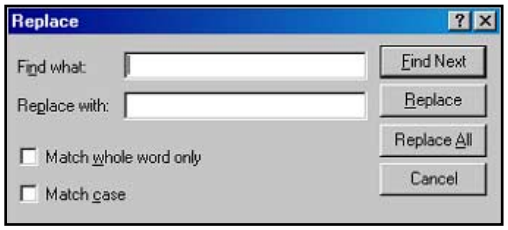
Select All: (Ctrl+A) Selects (highlights) the entire data in the presently active document, making it ready for global cut or copy operations to the windows clipboard.



__Find__ window

Find: (Ctrl+F) Displays a "**Find**" dialog to perform a text string search in currently active document (left).

Find Next: (F3) Repeats previous Find text search operation in the currently active document.



__Replace__ window

Replace: (Ctrl+H) Displays the **Replace** window to perform Find and Replace operations in the currently active document (left).

Find Next Error: Places the cursor at the next error in the currently active source file.

Find Previous Error: Places the cursor at the previous error in the currently active source.

Go To Top Of File: Places the cursor at the beginning of the currently active source file document.

Go To End Of File: (Ctrl+End) Places the cursor at the end of the currently active source file document.

Make SmartMotor Executable: (Ctrl+E) Scans source code within the currently active document for syntax and statement label errors and creates both uncompiled and compiled program files.

View Menu Commands:

Toolbar: Shows or hides the tool bar.

Status Bar: Shows or hides the status bar.

Communicate Menu Commands:

Communicate *menu commands*

Talk to SmartMotor(s): Opens the **SmartMotor Terminal** window, if it is not already open.

Address Motor Chain: Automatically addresses the motors connected to the serial port. This only works with an RS-232 Daisy Chain.

Monitor Status: Opens the **Monitor Status** window, if not already open and starts polling the default addressed SmartMotor.

Advanced Status: Opens the **Advanced Monitor Status** window, if not already open.

Set tuning: Displays the **Set Tuning Parameter** window.

Report Tuning: Requests the tuning parameters from default addressed SmartMotor and displays them in the Terminal window.

Send ECHO: Sends a global **ECHO** command to all SmartMotors. After this command is received, a SmartMotor receiving commands or data will echo it to the next motor or back to the terminal.

Send ECHO_OFF: Sends a global **ECHO_OFF** command to all SmartMotors. After this command is received, a SmartMotor receiving commands or data will not echo it to the next motor or back to the **SmartMotor Terminal.**

Send END: Sends a global **END** command to all SmartMotors.

Send RUN: Sends a global **RUN** command to all SmartMotors.

Send New Baud rate: Issues the new bit-rate to all SmartMotors and then changes the **SMI** bit-rate to the new value. The new bit-rate is displayed in the second pane of the **SmartMotor Terminal** ***Status bar***.

Report Variables:

Report **a…z**: Request default addressed SmartMotor to report the value of variables **a** through **z** (version 3.4 and earlier SmartMotors only have variables **a** through **j)**.

Report **aa…zz**: Request default addressed SmartMotor to report the value of variables **aa** through **zz** (not available for Version 3.4 and earlier SmartMotors).

Report **aaa…zzz**: Request default addressed SmartMotor to report the value of variables **aaa** through **zzz** (not available for Version 3.4 and earlier SmartMotors).

Report **ab[0]…ab[200]**: Request default-addressed SmartMotor to report the value of 8 bit variables **ab[0]** through **ab[200]** (not available for Version 3.4 and earlier SmartMotors).

Report **aw[0]…aw[99]**: Request default-addressed SmartMotor to report the value of 16 bit variables **aw[0]** through a**w[100]** (not available for Version 3.4 and earlier SmartMotors).

Report **al[0]…al[50]**: Request default-addressed SmartMotor to report the value of 32 bit variables **al[0]** through **al[50]** (not available for Version 3.4 and earlier SmartMotors).

List Macro Definitions:

List user macro definitions to the **SmartMotor Terminal** window.

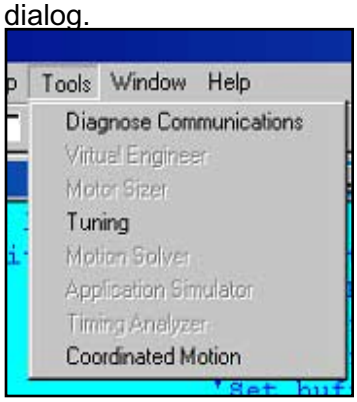
Setup Menu Commands:

Configure Host Port:

Displays the Set Default Communications Port dialog.

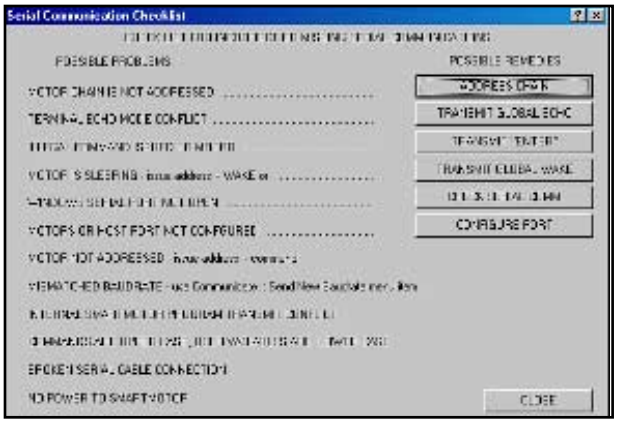
Tools Menu Commands:

Tools *drop-down menu commands*



Diagnose Communication: This command displays the **Serial Communications Checklist** window. If there is a communication problem with a SmartMotor, choose **Tools** from the menu bar and **Diagnose Communications** from the drop-down menu. This should turn on the **Serial Communication Checklist** window (below)

Serial Communication Checklist *window*



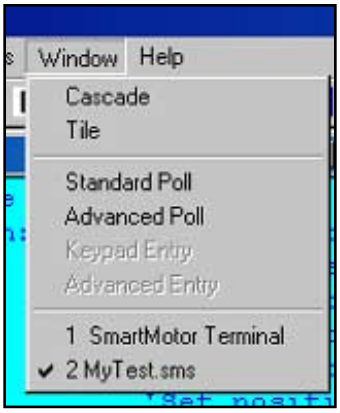
Read all of the lines on the left side and press the button to the right (if any) or do what is recommended.

Virtual Engineer: (future)

Motor Sizer: (future)

Tuning: The Tuning Utility is a stand-alone tool that can be used to plot the position of the shaft as it makes a quick

step motion. It has an integrated interface that allows the user to easily modify the **P.I.D.** Parameters and update them on-the-fly. The Tuning utility is detailed in a later section.



***Window** drop-down menu*

Motor Solver: (future)

Application Simulator: (future)

Timing Analyzer: (future)

Coordinated Motion: This is another stand-alone tool that manages the sending of coordinate data to SmartMotors.

Window Menu Commands

Cascade: Arranges the open document windows in overlapped fashion within the main program window.

Tile: Arranges the open windows in a non-overlapped fashion within the main program window.

Standard Poll: Brings the Standard Poll dialog into focus (useful if no mouse is in use).

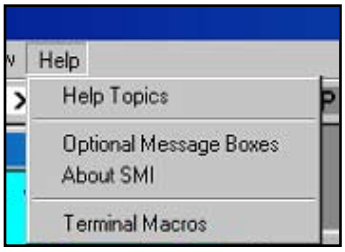
Advanced Poll: Brings the Advanced Poll dialog into focus (useful if no mouse is present).

Keypad Entry: Brings the Keypad Entry dialog into focus (useful if no mouse is present).

Advanced Entry: Brings Advanced Entry dialog into focus (useful if no mouse is present).

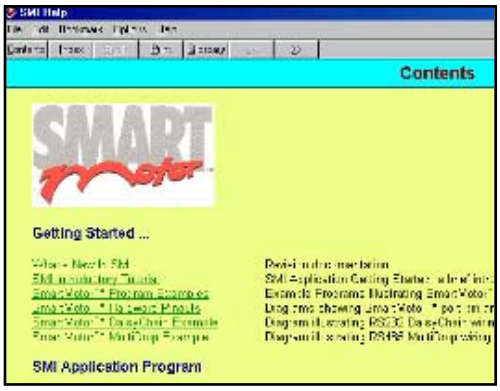
Help Menu Commands

Help Topics: Displays the **SMI** application help contents page.



***Help** drop-down menu*

Optional Message Boxes: The **SMI** application displays some message dialogs that the user may select not to show again. To allow the user to reset all option message dialogs to reappear, this command displays the **Reset All Message Dialogs** to open the window.



***Help** topics*

About SMI: Displays the About SMI dialog indicating the SMI version number.

Terminal Macros: Displays the Terminal Macros dialog window, a quick macro functions reminder.

Toolbar button description table

TOOLBAR COMMANDS

The following table summarizes the functions of the ***Toolbar*** buttons:

[illegible]	[illegible] Menu bar command	Description
	File -> New	Opens a new editing window.
	File -> Open	Opens a [illegible]
	File -> Save	Saves the currently [illegible]
	Edit -> Cut	[illegible]
	Edit -> Copy	[illegible]
	Edit -> Paste	[illegible] active document at current cursor position.
	File -> Print	Prints contents of currently active window.
	Edit -> Go To Top Of File	Places the cursor at the beginning of the [illegible]
	Edit -> Go To End Of File	[illegible] active document.
B	Edit -> [illegible]	[illegible]
	Edit -> Find Previous [illegible]	[illegible]
	Edit -> Find Next Error	[illegible]
T	[illegible]	[illegible]
	[illegible]	[illegible]
	[illegible]	[illegible]
	[illegible]	[illegible] SmartMotor.
	[illegible]	[illegible]
M	Communications -> Monitor [illegible]	[illegible]
P	[illegible] Advanced Status	[illegible]
S	[illegible]	[illegible]
	[illegible]	[illegible]
	[illegible]	[illegible]
	[illegible]	[illegible]

TUNING UTILITY OVERVIEW

The SmartMotor tuning is not done with the traditional turning of pots on an amplifier. These physical components have been replaced with firmware and host level software. The Tuning Utility is the host level component that aids in the selection of the absolute best values for optimal performance for a given loading condition. With this utility the values of **KP**, **KI**, **KD**, **KL**, **KS**, **KV**, **KA**, and **KG** can be changed and the motor's response to a step change in target position can be seen. A later section titled "The PID Filter" describes the functions of the different terms and the optimization process in detail.

NOTE:

The Tuning Utility instantly rotates the motor's shaft at maximum allowed acceleration and velocity for a quarter of a turn, to an abrupt stop. This can cause the motor to shake enough to affect delicate equipment.

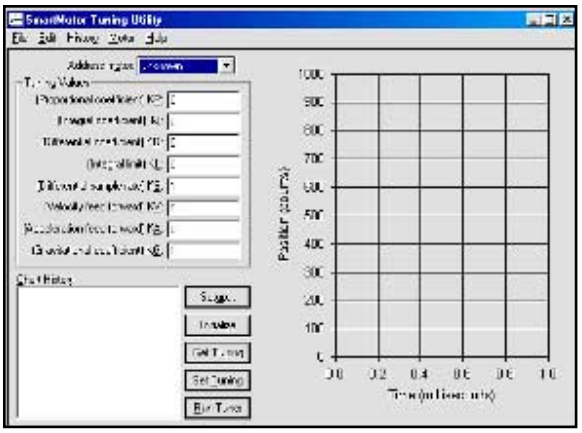
If the motor has an external Memory Module, remove it.

A QUICK TUTORIAL

This section is a quick guide for getting started using the **SmartMotor Tuning Utility** software.

Running the Tuner program

The **Tuning utility** was installed as part of the **SmartMotor Interface** software. To open the utility click on the Windows **Start** button and then click on **Programs**, **Animatics** and the **SmartMotor Tuning Utility**. The **Tuning Utility**'s main window, right, should now be on screen. The tuning utility can also be launched from the tools menu of the **SMI**.



*The SmartMotor **Tuning Utility**'s main window*

Setting up the Tuner

Click on the **Setup…** button

OR select **Tuner Setup…** from the **Motor** drop-down menu.

The **Setup** dialog box should now be on screen, right.

The Setup dialog box

Make sure the following default values are set in the dialog box:

Peak Velocity: 10000000
(The Maximum Velocity)

Peak Acceleration: 10000000 (The Maximum Acceleration)

Position Offset: 500 (The relative position change)

Number of Samples: 50 (Total number of samples taken)

Select the number of motors in the daisy chain from the **Max Motors** edit box and the communication port that is connected to the motors and leave all other check box entries in the window unchecked. Click on "**OK**" to set these values for tuning.

Initializing a Daisy chain of motors

If the motor(s) have an external Memory Module, remove it and make sure all connections are OK.

Click on the **Initialize** button

OR select **Initialize Daisy Chain** from the **Motor** drop-down menu

Selecting the motor after initializing.

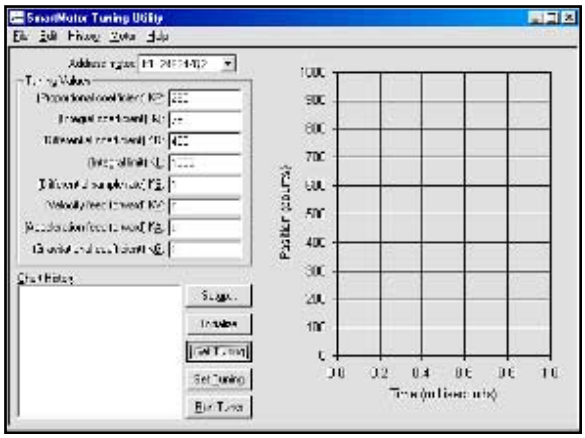
A small box shows that the program is initializing the motors in the daisy chain. After a few seconds the box will disappear and the number and version of the motor(s) should be visible in the **Address motor:** window. Select the motor from this combo box.

Getting the tuning values from a motor

Click on the **Get Tuning** button

OR select the **Get Tuning Values** from the **Motor** drop-down menu

Main window showing values read from motors.



This command asks the tuning values that are currently set in the motor. After a few seconds a message box appears showing the successful reading of values.

Setting the tuning values

Make sure that the values read from the motor are the motor's default values or are the correct values intended to be used in an application. The preceding figure displays the defaults for a version 4.02 motor. Modify the tuning values if necessary.

After verifying the values:

Click on the **Set Tuning** button

OR select the **Set Tuning Values** from the **Motor** drop-down menu

After a few seconds a message box shows the successful setting of motor values.

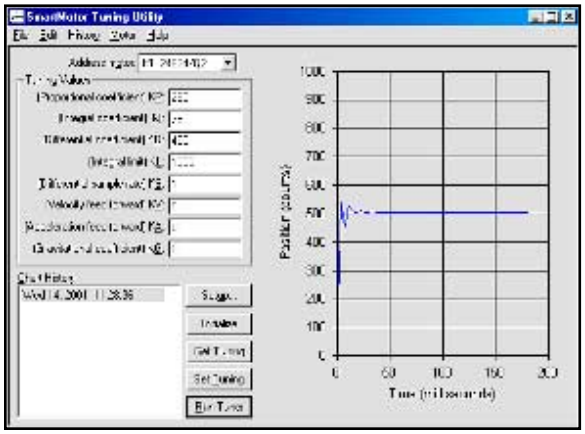
Running the Tuner

Now everything is ready to run the tuner.

Click on the **Run tuner** button

OR select the **Run tuner** from the **Motor** drop-down menu

A small box shows the program is processing the command and after a few seconds, the motor shaft will rotate rapidly and stop. After that, a record is added in the **Chart History** window, showing the date and time of operation and the results of the operation are shown in the chart on the right side of main window.



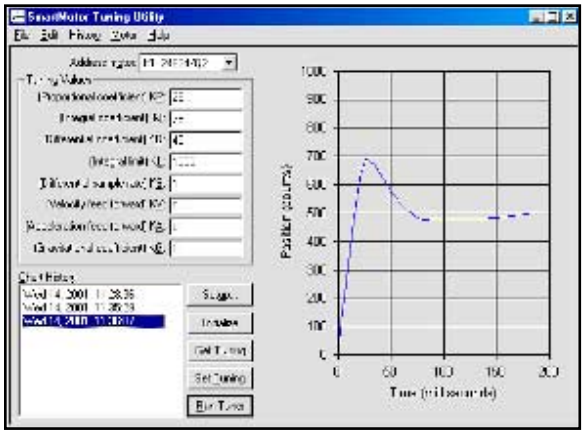
The resulting chart after running the tuner.

*See the **PID Filter** section for a greater understanding on optimizing the SmartMotor's tuning.*

Modifying the tuning values

Try other tuning values and setup parameters and compare the results.

Note: The **Set Tuning** button must be clicked each time the tuner is run. The figure, right, shows the result of running the tuner with a lowered **KP** and **KD**.



Running tuner with modified tuning values.

The date and time in the **Chart History** window is a default label that can be renamed by:

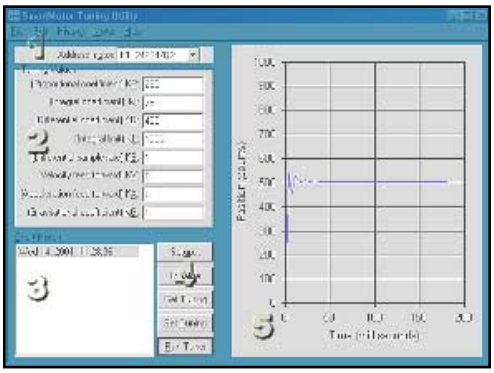
Clicking on a selected label

OR selecting **Edit** from the **History** pull-down menu and editing the label.

SMARTMOTOR TUNING UTILITY WINDOWS

There are five main elements of the **SmartMotor Tuning Utility** window.

The five elements of the "SmartMotor Tuning Utility".



1. **Address motor:** This window is updated after initialization of a single motor or a daisy chain of motors. There will be a line for each motor that can be accessed by clicking on the small down arrow at the end of the window. The data for each motor will show the sequence number, sample rate and version number. Highlight the motor needed and the tuner's commands will be sent to it.

*Note: The **PID** filter parameters are held in registers until an **F** command is received. Then they become active within the same servo cycle.*

2. **Tuning Values:** The Tuning values are 8 windows with data that defines the characteristics of a motor in response to different input data.

   (Proportional coefficient) **KP**: This is the gain of the proportional element in the **PID** filter. The higher the value of **KP**, the stiffer the motor will be.

   (Integral coefficient) **KI**: The integral compensation gain of the PID filter. The integral term of the **PID** filter creates a torque that is a function of both error and time. If the position error remains nonzero, over time, the torque becomes ever larger to enable the motor's shaft to reach its target.

   (Differential coefficient) **KD**: The derivative element of the **PID** filter. It can be thought of as the vibration-absorbing element.

   (Integral limit) **KL**: The integral limit of the **PID** filter. This value provides a limit to the amount of torque the **KI** term can produce given a non closing position error.

   (Differential sample rate) **KS**: This value represents the number of sample periods between evaluation of the **KD** parameter.

   (Velocity feed forward) **KV**: This value compensates for the predictable natural latency of the filter as it's influence grows with speed.

   (Acceleration feed forward) **KA**: This value compensates for the predictable forces due purely to acceleration and deceleration.

   (Gravitational coefficient) **KG**: This value compensates for the predictable force due to gravity in a vertical application.

***SmartMotor Tuning Utility** data can be saved with the file extension **.tnh**. Reopen the files in the **SmartMotor Tuning Utility** to view previously saved results.*

*Peak Velocity, Peak Acceleration, Position Offset, Number Of Samples, the Auto-scale option and the **PID** tuning values are all saved.*

3. **Chart History:** This window shows all of the tuning operations performed on the motor. Move through the list of motors (in step one) and see each tuning chart. Any item selected from this list can also be edited or deleted.

4. **Shortcut buttons:** Each one of these buttons has a corresponding sub-item in the "Motor" pull-down menu. The menu items operations are described in the following section.

5. **Tuning Chart:** This graph displays the "SmartMotor Tuning Utility" data for the selected SmartMotor. The horizontal axis is time in milliseconds and the vertical axis is position in counts.

SMARTMOTOR TUNING UTILITY MENUS

This section describes all of the operations performed by the **SmartMotor Tuning Utility.**

File, pull-down menu

The file pull-down menu (right) has 5 sub-items that are described here.

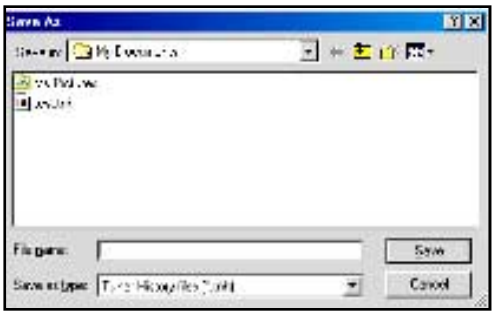
The File menu sub-items

New: Opens a new document. The document can be saved with a new file name.

Open: Opens an existing SmartMotor file. Select any file with **.tnh** extension saved in the **SmartMotor Tuning Utility**. All tuning values and tuning parameters are updated with the values stored in the file.

Save: Updates the data in the original saved file. If the data has not been saved (a new file), the **Save As** window (below) will open.

Save As: Opens the **Save As** window. Enter a new file name and save the **SmartMotor Tuning Utility** data to a new file.



*The **Save As** dialog box.*

Exit: Closes the **SmartMotor Tuning Utility**. If the file was modified since the last time it was saved, **Exit** will ask if the file should be saved.

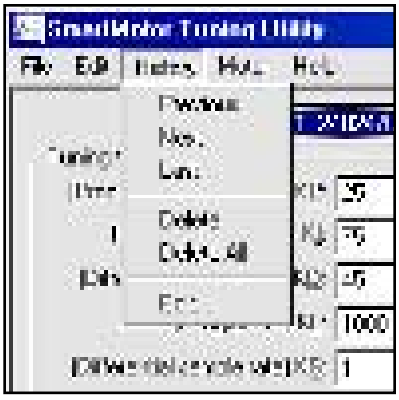
Edit pull-down menu

Copy: Will copy the data highlighted to the Windows clipboard. The data can then be pasted into another application such as a spreadsheet. It is also possible to paste a picture of the graph by using the **Paste Special** menu item in the destination application and selecting the Picture or Bit map format. The Picture format is scalable, while the Bit map format is not.

Use the SmartMotor version number (or other identification) in the filename as a reminder about which motor was used.

The data in the first column of numbers copied to the clipboard are the number of milliseconds since the start of the step motion. The second column contains the actual position values of the shaft at the corresponding times.

History pull-down menu



History menu sub-items.

The commands in this menu are related to the history of charts that are stored in the program.

Previous

This command selects the chart that is immediately before the current chart in the "Chart History" list box and shows it on the right side of main window.

Next

This command selects the chart that is immediately after the current chart in the "Chart History" list box and shows it on the right side of the main window.

Last

This command selects the last chart in the "Chart History" list box and shows it on the right side of the main window.

Delete

This command deletes the currently selected chart in the chart history list box.

Delete All

This command deletes all of the charts in the chart history.

Edit

This command allows the user to edit the **Chart History** label. This can also be accomplished by clicking on a selected item in the list. By default, the chart label indicates the date and time the chart was created. Click on a selected chart history label and edit the label to indicate any specific notes or comments about that chart. Click on **Enter** to complete editing or **Escape** to cancel editing.

Motor drop-down menu

***SmartMotor Tuning Utility/ Motor** drop-down menu*

This menu (right) contains sub-items that perform the main actions of the **SmartMotor Tuning Utility**.

Tuner Setup...

This command invokes the **Setup** dialog box (top of facing page).

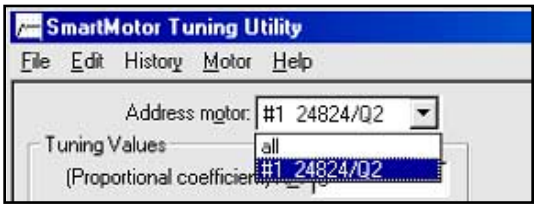
The parameters in this dialog box determine the data fed to motors for tuning. These values are as follows:

Setup dialog box.

Peak Velocity: is the target trajectory velocity that will be used in the step motion. The default value is 10 million.

Peak Acceleration: is the desired acceleration at which to rotate the shaft during the step motion. The default is 10 million. Due to limited resolution, odd numbers are rounded down to the next even integer.

Position Offset: specifies the target trajectory position. Note that nothing will happen if this value is too small. The default value is 500.



Sequence number, sample rate and version number identifiers in the combo box.

Number of Samples: is the number of times to repeat a loop that reads the position values. The amount of time spent in polling the position will vary with the speed of the computer. Currently, the minimum period between readings is about 3.3 milliseconds, sufficient for accurate rendering of the shaft position. The maximum limit is 1,000 samples, where the position polling will last more than 3 seconds at that value.

Max Motors: specifies the maximum number of motors to try and address before giving up. The larger the value, the longer the delay before the Tuner detects that the chain is not communicating or that there are no motors attached. It is not advisable to run the tuner in a daisy chain, because each motor adds a small amount of propagation delay (about 1-2ms). It also takes time to initialize a long chain.

Serial Port: where the motors are connected (COM1 or COM2).

Automatically initialize daisy chain on startup: This setting is only for convenience and generally should not be checked. It will cause the **Tuning Utility** to try to initialize a chain before loading its main window, which will introduce a small delay at startup. Use this option only if a motor will be connected before executing the **Tuner** application.

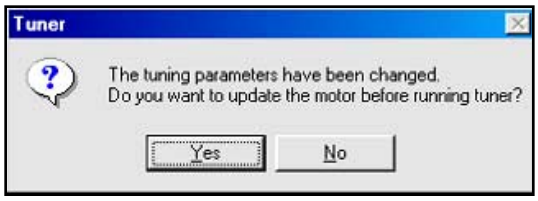
Window confirming tuning values have been successfully retrieved.

Automatically get tuning values: If this box is checked, the Tuner will retrieve the tuning values of the default motor immediately after initializing a daisy chain as well as every time a different Default Motor is selected in the pull-down box.

Automatically set tuning values: Enabling this setting will cause the tuning values entered in the dialog box to be sent to the Default Motor before running the tuner. Otherwise, if any value is modified before running the tuner and without setting those values in the motor, there will be prompt to update the tuning

Window confirming tuning values have been successfully set.

Window confirming motor update.



values in the motor with the newly edited values.

Auto-scale Y axis: The default scaling of the Y (position) axis is so that the target trajectory position offset is in the center of the axis. If there are any values that fall outside this preset range, turn this option on and the chart will auto-scale to fit the entire lower and upper bounds of the chart.

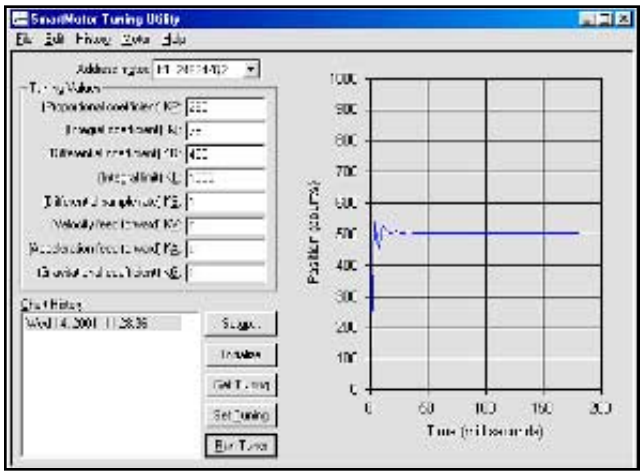
Initialize Daisy Chain: In order to use the Tuning Utility with more than one SmartMotor in a daisy chain, the daisy chain must first be initialized by selecting this menu item or pressing the "Initialize" button. With one motor, there's no need to initialize a daisy chain. The Address Motor combo box clears for a moment and then becomes filled with the sequence number, sample rate and version number of each motor in the daisy chain as shown.

Get Tuning Values: To examine the **PID** values stored in the motor, select this menu item or click on the **Get Tuning** button. This will display the tuning values stored in the currently addressed motor. A dialog box pops up to confirm that the tuning values have been successfully retrieved.

Set Tuning Values: Set the tuning values of the currently addressed motor by typing in new values for the **PID** terms and selecting this menu item or pressing the **Set Tuning** button. A confirmation message box pops up to indicate that the tuning values have been successfully set and the **F** command has been issued to load the values into active registers.

The resulting chart of running the tuner.

Run Tuner: Select this menu item or click on the **Run Tuner** button to run the motor and see the results. Depending on the tuning parameters, with no load attached to the motor, it may simply vibrate a little before settling down. A chart will plot, originating from the bottom left corner if the window and then oscillating up and down as shown here.



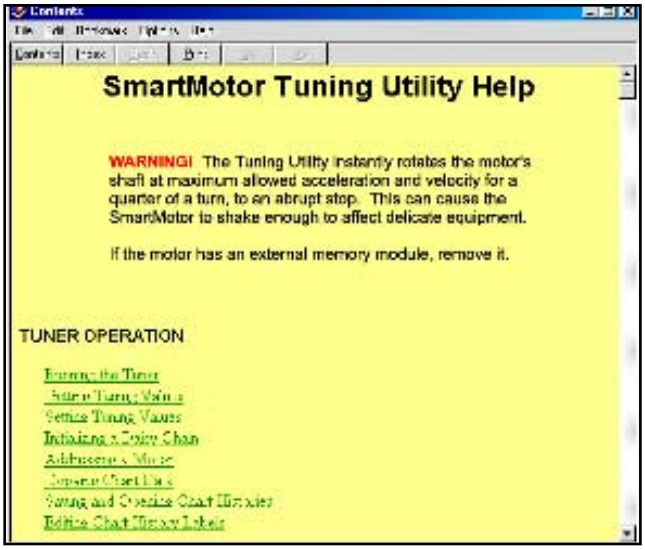
Help Menu:

The help menu has two sub-items.

Help menu.

Contents: This brings a dialog box showing the contents for the Smart Motor Tuning Utility help. All features of this program are described in this help system.

About…: This window gives information about the version and copyright.



The contents of SmartMotor Tuning Utility help.

PROGRAMMING TABLE OF CONTENTS

Enter the four commands below in the **SmartMotor Terminal** window, following each command with a return, and the SmartMotor will start to move:

Commands	Comments
A=100	'Set Maximum Acceleration
V=1000000	'Set Maximum Velocity
P=1000000	'Set Absolute Position
G	'Start move (Go)

A complete move requires the user to set a Position, a Velocity and an Acceleration, followed by a Go.

On power-up the motor defaults to position mode. Once **Acceleration** (**A**) and **Velocity** (**V**) are set, simply issue new **Position** (**P**) commands, followed by a **G** (**Go**) command to execute moves to new absolute locations. The motor does not instantly go to the programmed position, but follows a trajectory to get there. The trajectory is bound by the maximum **Velocity** and **Acceleration** parameters. The result is a trapezoidal velocity profile, or a triangular profile if the maximum velocity is never met.

Position, **Velocity** and **Acceleration** can be changed at any time during or between moves. The new parameters will only apply when a new **G** command is sent.

All SmartMotor commands are grouped by function, with the following notations:

#	Integer number
exp	Expression or signed integer
var	Variable
COM	Communication channel

A=exp Set absolute acceleration

Acceleration must be a positive integer within the range of 0 to 2,147,483,648. The default is zero forcing something to be entered to get motion. A typical value is 100. If left unchanged, while the motor is moving, this value will not only determine acceleration but also deceleration which will form a triangular or trapezoidal velocity motion profile. This value can be changed at any time. The value set does not get acted upon until the next **G** command is sent.

For SM17 & SM23
A=rev/sec^2 * 7.91

For SM34, 42 & 56
A=rev/sec^2 * 15.82

If the motor has a 2000 count encoder (sizes 17 and 23), multiply the desired acceleration, in rev/sec^2, by 7.91 to arrive at the number to set **A** to. With a 4000 count encoder (sizes 34, 42 and 56) the multiplier is 15.82. These constants are a function of the motors **PID** rate. If the **PID** rate is lowered, these constants must raised proportionally.

V=exp Set maximum permitted velocity

For SM17 & SM23
*V=rev/sec * 32212*

For SM34, 42 & 56
*V=rev/sec * 64424*

Use the **V** command to set a limit on the velocity the motor can accelerate to. That limit becomes the slew rate for all trajectory based motion whether in position mode or velocity mode. The value defaults to zero so it must be set before any motion can take place. The new value does not take effect until the next **G** command is issued. If the motor has a 2000 count encoder (sizes 17 and 23), multiply the desired velocity in rev/sec by 32212 to arrive at the number to set **V** to. With a 4000 count encoder (sizes 34, 42 & 56) the multiplier is 64424. These constants are a function of the motors **PID** rate. If the **PID** rate is lowered, these constants will need to be raised.

P=exp Set absolute position for move

For SM17 & SM23
*P=rev * 2000*

For SM34, 42 & 56
*P=rev * 4000*

The **P=** command sets an absolute end position. The units are encoder counts and can be positive or negative. The end position can be set or changed at any time during or at the end of previous moves. SmartMotor sizes 17 and 23 resolve 2000 increments per revolution while SmartMotor sizes 34, 42 and 56 resolve 4000 increments per revolution.

The following program illustrates how variables can be used to set motion values to real-world units and have the working values scaled for motor units for a size 17 or 23 SmartMotor.

```
a=100           'Acceleration in rev/sec*sec
v=1             'Velocity in rev/sec
p=100           'Position in revs
GOSUB10         'Initiate motion
END             'End program
C10             'Motion routine
   A=a*8        'Set Acceleration
   V=v*32212    'Set Velocity
   P=p*2000     'Set Position
   G            'Start move
RETURN          'Return to call
```

D=exp Set relative distance for position move

The **D=** command allows a relative distance to be specified, instead of an absolute position. The number following is encoder counts and can be positive or negative.

The relative distance will be added to the current position, either during or after a move. It is added to the desired position rather than the actual position so as to avoid the accumulation of small errors due to the fact that any servo motor is seldom exactly where it should be at any instant in time.

G Go, start motion

G also resets several system state flags

The **G** command does more than just start motion. It can be used dynamically during motion to create elaborate profiles. Since the SmartMotor allows position, velocity and acceleration to change during motion, “on-the-fly”, the **G** command can be used to trigger the next profile at any time.

S Abruptly stop motion in progress

If the **S** command is issued while a move is in progress it will cause an immediate and abrupt stop with all the force the motor has to offer. After the stop, assuming there is no position error, the motor will still be servoing. The **S** command works in both Position and Velocity modes.

X Decelerate to stop

If the **X** command is issued while a move is in progress it will cause the motor to decelerate to a stop at the last entered **A=** value. When the motor comes to rest it will servo in place until commanded to move again. The **X** command works in both Position and Velocity modes.

O=exp Set/Reset origin to any position

The **O=** command (using the letter **O**, not the number zero) allows the host or program not just to declare the current position zero, but to declare it to be any position, positive or negative. The exact position to be re-declared is the ideal position, not the actual position which may be changing slightly due to hunting or shaft loading. The **O=** command directly changes the motor's position register and can be used as a tool to avoid +/- 31 bit roll over position mode problems. If the SmartMotor runs in one direction for a very long time it will reach position +/-2,147,483,648 which will cause the position counter to change sign. While that is not an issue with Velocity Mode, it can create problems in position mode.

OFF Turn motor servo off

The **OFF** command will stop the motor from servoing, much as a position error or limit fault would. When the servo is turned off, one of the status LEDs will revert from Green to Red.

MP Position Mode

Position mode is the default mode of operation for the SmartMotor. If the mode were to be changed, the **MP** command would put it back into position mode. In position mode, the **P#** and **D#** commands will govern motion.

BINARY POSITION DATA TRANSFER

The **ASCII** based command string format, while convenient, is not the fastest way to communicate data. It can be burdensome when trajectory commands are sent to the motor. For that reason a special binary format has been established for the communication of trajectory critical data such as **Position**, **Velocity** and **Acceleration**. Using the binary format, these 32 bit parameters are sent as four bytes following a code byte that flags the data for a particular purpose. The code bytes are 252 for acceleration, 253 for velocity and 254 for position. As an example, the following byte values communicate A=53, V=-1 & P=2137483648.

A=53	252 000 000 000 053 032
V=-1	253 255 255 255 254 032
P=2137483648	254 127 255 255 255 032

For further expediency, the commands can be appended with the **G** command to start motion immediately. Two examples are as follows (the ASCII value for **G** is 71):

P=0 G	254 000 000 000 000 071 032
V=512 G	253 000 000 002 000 071 032

MV Velocity Mode

Velocity mode will allow continuous rotation of the motor shaft. In Velocity mode, the programmed position using the **P** or the **D** commands is ignored. Acceleration and velocity need to be specified using the **A=** and the **V=** commands. After a **G** command is issued, the motor will accelerate up to the programmed velocity and continue at that velocity indefinitely. In velocity mode as in Position mode, Velocity and Acceleration are changeable on-the-fly, at any time. Simply specify new values and enter another **G** command to trigger the change. In Velocity mode the velocity can be entered as a negative number, unlike in Position mode where the location of the target position determines velocity direction or sign. If the 32 bit register that holds position rolls over in velocity mode it will have no effect on the motion.

MT Torque Mode

In torque mode the motor shaft will simply apply a torque independent of position. The internal encoder tracking will still take place, and can be read by a host or program, but the value will be ignored for motion because the **PID** loop is inactive. To specify the amount of torque, use the **T=** command, followed by a number between -1023 and 1023.

T=exp Set torque value, -1023 to 1023

In torque mode, activated by the **MT** command, the drive duty cycle can be

set with the **T=** command. The following number or variable must fall in the range between -1023 and 1023. The full scale value relates to full scale or maximum duty cycle. At a given speed there will be reasonable correlation between drive duty cycle and torque. With nothing loading the shaft, the **T=** command will dictate open-loop speed.

MD Contouring Mode (requires host)

Contouring Mode is the foundation of Animatics' G-Code interface that enables a P.C. and multiple SmartMotors to interpret G-Code files and do multiple axis contouring.

SmartMotors with version 4.15 or greater firmware have the added ability to do multiple axis contouring. This firmware version became standard roughly mid-year 2001. The **Contouring Mode** is the foundation of the Animatics' G-Code interface that enables a P.C. and multiple SmartMotors to interpret G-Code files and do linear, circular and helical interpolation as well as unlimited multi-axis contouring.

The basic principle of operation takes advantage of the fact that each SmartMotor has a very accurate time base. Absolute position-time pairs of data get sent to the SmartMotor to fill buffers that facilitate continuous motion. The SmartMotor will adjust its own Velocity and Acceleration to be certain to arrive at the specified position at the exact specified time without slowing to a stop. As new position-time pairs arrive, the motor transitions smoothly from one profile to the next producing smooth, continuous motion. In a multiple axis configuration, different positions can be sent to different motors, with the same time intervals resulting in smooth, continuous multiple axis motion. The key is for the host to regulate the volume of data in each of the different motor's buffers. The position-time pairs of data are preceded with an identification byte and then four bytes for position and four for time. Time is in units of servo samples and is limited to 23 bits. Time is further constrained to be even powers of 2 (i.e. 1, 2, 4, 8, ..., 32768).

The coordinating host can send the **Q** command to solicit status information on the coordination process. Upon receiving the **Q** command, the SmartMotor will return status, clock and space available in the dedicated circular buffer. The response to **Q** takes two forms, one while the mode is running with trajectory in progress and no errors having occurred and another when the mode is not running. Both responses conform to the overall byte format of:

Q Response: 249 byte1 byte2 byte3 byte4

If the mode is running:

byte1 bit 7 is set
byte1 bits 6 through 0 return data slots available
bytes 2, 3 & 4 return the 24 bit clock of the SmartMotor

If the mode is not running:

byte1 bit 7 is clear
byte1 bits 6 through 0 return status
byte2 returns space available
bytes 3 & 4 return the 12 lower bits of the 24 bit clock of the SmartMotor

As absolute position and time data is sent to the SmartMotor, differences are calculated that we call "deltas". A delta is the difference between the latest value and the one just prior. Time deltas are limited to 16 bits while Position deltas are limited to 23 bits in size.

The Status Byte is constructed as follows:

bit0=1	MD mode pending a G
bit1=1	MD mode actually running
bit2=1	Invalid time delta > 16 bit received
bit3=1	Invalid position delta > 23 bits received
bit4=1	Internal program data space error
bit5=1	Host sent too much data (data buffer overflow)
bit6=1	Host sent too little data (data buffer underflow)

A trajectory terminates if an unacceptable position error occurs, if invalid data is received, if there is a data overflow or if there is a data underflow.

The host should send data pairs only when at least 3 empty data slots are available. **MD** responds to limit switches with an aborted trajectory. The **MD** mode uses KV feed forward for improved performance.

The byte flag that precedes and marks a position is of decimal value 250. The byte flag that precedes and marks a time is of decimal value 251.

The following is an example of the decimal byte values for a series of constant speed motion segments. Firmware versions 4.16 and higher do not need time values after the first two if the time delta is not changing. The byte transfers terminate with a carriage return (13).

Position	250 000 000 000 000 013	Position = 0
Time	251 000 000 000 000 013	Time = 0
Position	250 000 000 016 000 013	
Time	251 000 000 001 000 013	Time delta = 256
Position	250 000 000 032 000 013	
Position	250 000 000 048 000 013	
Position	250 000 000 052 000 013	Reduce position delta
Time	251 000 000 003 064 013	Reduce time delta
Position	250 000 000 056 000 013	
Position	250 000 000 060 000 013	
Position	250 000 000 064 000 013	

What is not shown in the these codes are the addressing bytes that would be used to differentiate multiple motors on a network. As described ahead in this manual (see the **SADDR** command), a network of SmartMotors can

be sorted out by sending a single address byte. When communicating to a particular motor, the address byte need only be sent once, until all of the communications to that particular motor are complete and another motor needs to be addressed. The byte patterns in the previous example would need to be preceded with an address byte (to a properly addressed motor) for multiple axis contouring. In the addressing scheme, there is a global address provision for sending data to all motors at once. By zeroing out the clocks before starting the contouring, the motors will by synchronized and single time values can then be sent to all motors at once, increasing overall bandwidth. Also, as mentioned earlier, SmartMotors with version 4.16 or higher do not need time data past the first two, if there is no change in the time delta.

The basis for contouring using this format is to keep the rate at which data is sent to each motor constant (and as fast as possible). That means that in order to accelerate axes, absolute positions need to be sent that invoke progressively larger position deltas, and to keep constant velocity, absolute positions need to be sent that are equidistant.

With all of the communications to send data and receive status, it would be outstanding to have a bandwidth on a two axis system of 64 samples, or 16ms. Typically, with a three or four axis system a bandwidth of 128 servo samples or 32ms is achievable. This would be at a baud rate of 38.4k. Keep in mind that during this time the SmartMotor is micro interpolating. The motion will be very smooth and continuous.

In contouring mode, all of the binary contouring data goes into the motor's buffers. While this is true, regular commands will still be recognized and they will operate normally. This will take some time, however, and it is up to the programmer to assure that the buffers never underflow due to neglect.

BRAKE COMMANDS
(where optional brake exists)

BRKRLS Brake release

BRKENG Brake engage

BRKSRV Release brake when servo active, engage break when inactive.

BRKTRJ Release brake when running a trajectory, engage under all other conditions. Turns servo off when the brake is engaged.

Many SmartMotors™ are available with power safe brakes. These brakes will apply a force to keep the shaft from rotating should the SmartMotor lose power. Issuing the **BRKRLS** command will release the brake and **BRKENG** will engage it. There are two other commands that initiate automated operating modes for the brake. The command **BRKSRV** engages the brake automatically, should the motor stop servoing and holding position for any reason. This might be due to loss of power or just a position error, limit fault, over-temperature fault.

Finally, the **BRKTRJ** command will engage the brake in response to all of the previously mentioned events, plus any time the motor is not performing a trajectory. In this mode the motor will be off, and the brake will be holding it in position, perfectly still, rather than the motor servoing when it is at rest. As soon as another trajectory is started, the brake will release. The time it takes for the brake to engage and release is on the order of only a few milliseconds.

The brakes used in SmartMotors™ are zero-backlash devices with extremely long life spans. It is well within their capabilities to operate interactively within an application. Care should be taken not to create a situation where the brake will be set repeatedly during motion. That will reduce the brake's life.

Program commands are like chores, whether it is to turn on an output, set a velocity or start a move. A program is a list of these chores. When a programmed SmartMotor is powered-up or its program is reset with the **Z** command, it will execute its program from top to bottom, with or without a host P.C. Connected. This section covers the commands that control the program itself.

RUN Execute stored user program

If the SmartMotor is reset with a **Z** command, all previous variables and mode changes will be erased for a fresh start and the program will begin to execute from the top. Alternatively the **RUN** command can be used to start the program, in which case the state of the motor is unchanged and its program will be invoked.

RUN? Halt program if no RUN issued

To keep a downloaded program from executing at power-up start the program with the **RUN?** Command. It will prevent the program from starting when power is applied, but it will not prevent the program from running when the SmartMotor sees a **RUN** command from a host over the RS-232 port.

Once the program is running, there are a variety of commands that can redirect program flow and most of those can do so based on certain conditions. How these conditional decisions are setup determines what the programmed SmartMotor will do, and exactly how “smart” it will actually be.

GOTO# Redirect program flow

C# Subroutine label, C0-C999

The most basic commands for redirecting program flow, without inherent conditions, are **GOTO#** in conjunction with **C#**. Labels are the letter **C** followed by a number (**#**) between 0 and 999 and are inserted in the program as place markers. If a label, **C1** for example, is placed in a program and that same number is placed at the end of a **GOTO** command, **GOTO1**, the program flow will be redirected to label **C1** and the program will proceed from there.

As many as a thousand labels can be used in a program (0 - 999), but, the more **GOTO** commands used, the harder the code will be to debug or read. Try using only one and use it to create the infinite loop necessary to keep the program running indefinitely, as some embedded programs do. Put a **C1** label near the beginning of the program, but after the initialization code and a **GOTO1** at the end and every time the **GOTO1** is reached the program will loop back to label **C1** and start over from that point until the **GOTO1** is reached, again, which will start the process at

C1 again, and so on. This will make the program run continuously without ending. Any program can be written with only one **GOTO**. It might be a little harder, but it will tend to force better program organization, which in turn, will make it easier to be read and changed.

END End program execution

If it's necessary to stop a program, use an **END** command and execution will stop at that point. An **END** command can also be sent by the host to intervene and stop a program running within the motor. The SmartMotor program is never erased until a new program is downloaded. To erase the program in a SmartMotor, download only the **END** command as if it were a new program and that's the only command that will be left on the SmartMotor until a new program is downloaded. To compile properly, every program needs and **END** somewhere, even if it is never reached. If the program needs to run continuously, the **END** statement has to be outside the main loop.

GOSUB# Execute a subroutine

RETURN Return from subroutine

Just like the **GOTO#** command, the **GOSUB#** command, in conjunction with a **C#** label, will redirect program execution to the location of the label. But, unlike the **GOTO#** command, the **C#** label needs a **RETURN** command to return the program execution to the location of the **GOSUB#** command that initiated the redirection. There may be many sections of a program that need to perform the same basic group of commands. By encapsulating these commands between a **C#** label and a **RETURN**, they may be called any time from anywhere with a **GOSUB#**, rather than being repeated in their totality over and over again. There can be as many as one thousand different subroutines (0 - 999) and they can be accessed as many times as the application requires.

Calling subroutines from the host can crash the stack

By pulling sections of code out of a main loop and encapsulating them into subroutines, the main code can also be easier to read. Organizing code into multiple subroutines is a good practice.

The commands that can conditionally direct program flow to different areas use a constant **[#]** like 1 or 25, a variable like **a** or **al[#]** or a function involving constants and/or variables **a+b** or **a/[#]**. Only one operator can be used in a function. The following is a list of the operators:

- **+** Addition
- **-** Subtraction
- ***** Multiplication
- **/** Division
- **==** Equals (use two =)

!= Not equal

< Less than

> Greater than

<= Less than or equal

>= Greater than or equal

& Bit wise AND (see appendix A)

| Bit wise OR (see appendix A)

WHILE, LOOP

The most basic looping function is a **WHILE** command. The **WHILE** is followed by an expression that determines whether the code between the **WHILE** and the following **LOOP** command will execute or be passed over. While the expression is true, the code will execute. An expression is true when it is non-zero. If the expression results in a "zero" then it is false. The following are valid **WHILE** structures:

```
WHILE 1      '1 is always true
   UA=1      'Set output to 1
   UA=0      'Set output to 0
LOOP         'Will loop forever

a=1          'Initialize variable 'a'
WHILE a      'Starts out true
   a=0       'Set 'a' to 0
LOOP         'This never loops back

a=0          'Initialize variable 'a'
WHILE a<10   'a starts less
   a=a+1     'a grows by 1
LOOP         'Will loop back 10x
```

The task or tasks within the **WHILE** loop will execute as long as the function remains true.

The **BREAK** command can be used to break out of a **WHILE** loop, although that somewhat compromises the elegance of a **WHILE** statement's single test point, making the code a little harder to follow. The **BREAK** command should be used sparingly or preferably not at all in the context of a **WHILE**.

If it's necessary for a portion of code to execute only once based on a certain condition then use the **IF** command.

IF, ENDIF

Once the execution of the code reaches the **IF** command, the code between that **IF** and the following **ENDIF** will execute only when the condition directly following the **IF** command is true. For example:

```
a=UAI          'Variable 'a' set 0,1
a=a+UBI        'Variable 'a' 0,1,2
IF  a==1       'Use double = test
    b=1        'Set 'b' to one
ENDIF          'End IF
```

Variable **b** will only get set to one if variable **a** is equal to one. If **a** is not equal to one, then the program will continue to execute using the command following the **ENDIF** command.

Notice also that the SmartMotor language uses a single equal sign (**=**) to make an assignment, such as where variable **a** is set to equal the logical state of input **A**. Alternatively, a double equal (**==**) is used as a test, to query whether **a** is equal to 1 without making any change to **a**. These are two different functions. Having two different syntaxes has farther reaching benefits.

ELSE, ELSEIF

The **ELSE** and **ELSEIF** commands can be used to add flexibility to the **IF** statement. If it were necessary to execute different code for each possible state of variable **a**, the program could be written as follows:

```
a=UAI          'Variable 'a' set 0,1
a=a+UBI        'Variable 'a' 0,1,2
IF a==0        'Use double '=' test
   b=1         'Set 'b' to one
ELSEIF a==1
   c=1         'Set 'c' to one
ELSEIF a==2
   c=2         'Set 'c' to two
ELSE           'If not 0 or 1
   d=1         'Set 'd' to one
ENDIF          'End IF
```

There can be many **ELSEIF** statements, but at most one **ELSE**. If the **ELSE** is used, it needs to be the last statement in the structure before the **ENDIF**. There can also be **IF** structures inside **IF** structures. That's called “nesting” and there is no practical limit to the number of structures that can nest within one another.

SWITCH, CASE, DEFAULT, BREAK, ENDS

Long, drawn out **IF** structures can be cumbersome, however, and burden the program visually. In these instances it can be better to use the **SWITCH** structure. The following code would accomplish the same thing as the previous program:

The SWITCH statement makes use of the same memory space as variable "zzz". Do not use this variable or array space when using SWITCH

```
a=UAI          'Variable 'a' set 0,1
a=a+UBI        'Variable 'a' 0,1,2
SWITCH a       'Begin SWITCH
  CASE 0
    b=1        'Set 'b' to one
    BREAK
  CASE 1
    c=1        'Set 'c' to one
    BREAK
  CASE 2
    c=2        'Set 'c' to two
    BREAK
  DEFAULT        'If not 0 or 1
    d=1        'Set 'd' to one
    BREAK
ENDS           'End SWITCH
```

Just as a rotary switch directs electricity, the **SWITCH** structure directs the flow of the program. The **BREAK** statement then jumps the code execution to the code following the associated **ENDS** command. The **DEFAULT** command covers every condition other than those listed. It is optional.

TWAIT Wait during trajectory

The **TWAIT** command pauses program execution while the motor is moving. Either the controlled end of a trajectory, or the abrupt end of a trajectory due to an error, will terminate the **TWAIT** waiting period. If there were a succession of move commands without this command, or similar waiting code between them, the commands would overtake each other because the program advances, even while moves are taking place. The following program has the same effect as the **TWAIT** command, but has the added virtue of allowing other things to be programmed during the wait, instead of just waiting. Such things would be inserted between the two commands.

```
WHILE Bt       'While trajectory
LOOP           'Loop back
```

WAIT=exp Wait (exp) sample periods

*For the exact sample period, use the **RSP** command*

There will probably be circumstances where the program execution needs to be paused for a specific period of time. Time, within the SmartMotor, is tracked in terms of servo sample periods. Unless otherwise programmed with the **PID#** command, the sample rate is about 4KHz. **WAIT=**4000 would wait about one second. **WAIT=**1000 would wait for about one quarter of a second. The following code would be the same as **WAIT=**1000, only it will allow code to execute during the wait if it is placed between the **WHILE** and the **LOOP**.

```
CLK=0              'Reset CLK to 0
WHILE CLK<1000     'CLK will grow
  IF UAI==0        'Monitor input A
    GOSUB911       'If input low
  ENDIF            'End the IF
LOOP               'Loop back
```

The above code example will check if port **A** ever goes low, while it is waiting for the **CLK** variable to count up to 1000.

STACK Reset the GOSUB return stack

The **STACK** is where information is held with regard to the nesting of subroutines (nesting is when one or more subroutines exist within others). In the event program flow is directed out of one or more nested subroutines, without executing the included **RETURN** commands, the stack will be corrupted. The **STACK** command resets the stack with zero recorded nesting. Use it with care and try to build the program without requiring the **STACK** command.

One possible use of the **STACK** command might be if the program used one or more nested subroutines and an emergency occurred, the program or operator could issue the **STACK** command and then a **GOTO** command which would send the program back to a label at the beginning. Using this method instead of a **RESET** command would retain the states of the variables and allow further specific action to resolve the emergency.

Variables are data holders that can be set and changed within the program or over the communication channel.

The first 26 variables are long integers (32 bits) and are accessed with the lower case letters of the alphabet, **a, b**, **c**, . . . **x**, **y**, **z**.

a=# Set variable **a** to a numerical value

a=exp Set variable **a** to value of an expression

A variable can be set to an expression with only one operator and two operands. The operators can be any of the following:

+ Addition

- Subtraction

***** Multiplication

/ Division

& Bit wise AND (see appendix A)

| Bit wise OR (see appendix A)

See Appendix C for a table describing User Assigned Variables.

The following are legal:

a=b+c, a=b+3 a=5+8

a=b-c a=5-c a=b-10

a=b*c a=3*5 a=c*3

a=b/c a=b/2 a=5/b

a=b&c a=b&8

a=b|c a=b|15

ARRAYS

In addition to the first 26, there are 52 more long integer variables accessible with double and triple lower case letters: **aa, bb, cc, . . . xxx, yyy, zzz**. The memory space that holds these 52 variables is more flexible, however. This same variable space can be accessed with an array variable type. An array variable is one that has a numeric index component that allows the numeric selection of which variable a program is to access. This memory space is further made flexible by the fact that it can hold 51 thirty two bit integers, or 101 sixteen bit integers, or 201 eight bit integers (all signed).

The array variables take the following form:

ab[i]=exp Set variable to a signed 8 bit value where index i = 0...200

aw[i]=exp Set variable to a signed 16 bit value where index i = 0...100

al[i]=exp Set variable to a signed 32 bit value where index i = 0...50

The index **i** may be a number, a variable **a** thorough **z**, or the sum or difference of any two variables **a** thorough **z** (variables only).

The same array space can be accessed with any combination of variable types. Just keep in mind how much space each variable takes. We can even go so far as to say that one type of variable can be written and another read from the same space. For example, if the first four eight bit integers are assigned as follows:

ab[0]=0

ab[1]=0

ab[2]=1

ab[3]=0

They would occupy the same space as the first single 32 bit number, and due to the way binary numbers work, would make the thirty two bit variable equal to 256. The order is most significant to least with ab[0] being the most.

A common use of the array variable type is to set up what is called a buffer. In many applications, the SmartMotor will be tasked with inputting data about an array of objects and to do processing on that data in the same order, but not necessarily at the same time. Under those circumstances it may be necessary to “buffer” or “store” that data while the SmartMotor processes it at the proper times.

To set up a buffer the programmer would allocate a block of memory to it, assign a variable to an input pointer and another to an output pointer. Both pointers would start out as zero and every time data was put into the buffer the input pointer would increment. Every time the data was used, the output buffer would likewise increment. Every time one of the pointers is incriminated, it would be checked for exceeding the allocated memory space and rolled back to zero in that event, where it would continue to increment as data came in. This is a first-in, first-out or “FIFO” circular buffer. Be sure there is enough memory allocated so that the input pointer never overruns the output pointer.

STORAGE OF VARIABLES

(Not available in SMXXX5 SmartMotors™)

Newer SmartMotors have 32K of non-volatile EEPROM memory to store variables when they need to survive the motor powering down.

EPTR=expression Set EEPROM pointer, 0-7999

To read or write into this memory space it is necessary to properly locate the pointer. This is accomplished by setting **EPTR** equal to the offset.

VST(variable,index) Store variables

To store a series of variables, use the **VST** command. In the "variable" space of the command put the name of the variable and in the "index" space put the total number of sequential variables that need to be stored. Enter a one if just the variable specified needs to be stored. The actual sizes of the variables will be recognized automatically.

VLD(variable,index) Load variables

To load variables, starting at the pointer, use the **VLD** command. In the "variable" space of the command put the name of the variable and in the "index" space put the number of sequential variables to be loaded.

*See **Appendix C** for a table describing **User Assigned Variables.***

FIXED OR PRE-ASSIGNED FUNCTIONS

In addition to the general purpose variables there are variables that are gateways into the different functions of the motor itself.

@P	Current position
@PE	Current position error
@V	Current velocity
ADDR	Motor's self address
CHN0	RS-232 com error flags
CHN1	RS-485 com error flags
CLK	Read/Write sample rate counter
CTR	External encoder count variable
I	Last recorded index position
LEN	# of characters in RS-232 buffer
LEN1	# of characters in RS-485 buffer

REPORT TO HOST COMMANDS

Ra...**Rzzz**	Report variables **a** ... **zzz,** 78 in all
Rab[i]	Report 8 bit variable value Rab[i]
Raw[i]	Report 16 bit variable value Raw[i]
Ral[i]	Report 32 bit variable value Ral[i]
RA	Report buffered acceleration
RAIN{port}{ch}	Report 8 bit analog input port=A-H, ch= 1-4
RAMPS	Report assigned maximum current
RBa	Report over current status bit
RBb	Report parity error status bit
RBc	Report communications error bit
RBd	Report user math overflow status bit
RBe	Report position error status bit
RBf	Report communications framing error status bit
RBk	Report EEPROM read/write status bit
RBl	Report historical left limit status bit
RBi	Report index status bit
RBh	Report overheat status bit
RBm	Report negative limit status bit
RBo	Report motor off status bit
RBp	Report positive limit status bit
RBr	Report historical right limit status bit
RBs	Report program scan status bit
RBt	Report trajectory status bit
RBu	Report user array index status bit
RBw	Report wrap around status bit
RBx	Report hardware index input level
RCHN	Report combined communications status bits
RCHN0	Report RS-232 communications status bits
RCHN1	Report RS-485 communications status bits
RCLK	Report clock value
RCTR	Report secondary counter
RCS	Report RS-232 communications check sum
RCS1	Report RS-485 communications check sum
RD	Report buffered move distance value

RDIN{port}{ch}	Report 8 bit digital input byte, port=A-H, and ch=0-63
RE	Report buffered maximum position error
RI	Report last stored index position
RKA	Report buffered acceleration feed forward coefficient
RKD	Report buffered derivative coefficient
RKG	Report buffered gravity coefficient
RKI	Report buffered integral coefficient
RKL	Report buffered integral limit value
RKP	Report buffered proportional coefficient
RKS	Report buffered sampling interval
RKV	Report buffered velocity feed forward coefficient
RP	Report measured position
RPE	Report present position error
RMODE	Report present positioning mode: **P** Absolute position move **R** Relative position move **V** Velocity move **T** Torque mode **F** Follow mode **S** Step and Direction mode **C** Cam Table mode **W** Drive mode **X** Follow mode with multiplier **E** Position error **O** Motor off **H** Contouring mode
RS2	Restore RS-232 mode
RS4	Assign **UG** to RS-485 control
RS	Report status byte (8 system states)
RSP	Report sample period and version number
RT	Report current requested torque
RV	Report velocity
RW	Report status word (16 system states)

*See **Appendix C** for a table describing **User Assigned Variables.***

Through the two pins, A and B of the I/O connector, quadrature or step and direction signals can be fed into the SmartMotor at high speeds and be followed by the motor itself. This feature brings about the following capabilities:

1 Mode Follow

2 Mode Step and Direction

3 Mode Follow with ratio

4 Mode Step and Direction with ratio

5 Mode Cam

SmartMotors SMXXX5 do not have Quadrature Encoder following capability built-in, but can be adapted to accept quadrature.

In addition to the above embedded modes of operation, the internal counter can be set to either count encoder signals or step signals and be accessible to the internal program or a host through the **CTR** variable.

When the SmartMotor is in one of the above five modes it may also run internal programs and communicate with a host, all at the same time.

MF1, MF2 and MF4 Mode Follow

Mode Follow allows the SmartMotor™ to follow an external encoder. Three resolutions can be selected through hardware, and a virtually infinite number of resolutions can be set in firmware using the MFR command described ahead. Set the hardware for maximum resolution with the **MF4** command. The **MF2** The **MF1** commands set the hardware to lesser resolutions, but are obsolete with the advent of the newer MFR capability.

MF0, MS0

The **MF0** and **MS0** commands must not be issued during one of the other follow modes. They are used for an entirely different purpose. If it is not desired to directly follow an incoming encoder or step signal, but rather, just to track them and use the counter value within a program or from a host, then issuing **MF0** or **MS0** utilizes the maximum resolution available and makes the value available through the **CTR** variable. Issuing **MF0** or **MS0** will zero that variable and incoming encoder or step signals will increment or decrement the signed, 32-bit **CTR** variable value.

MFDIV=expression **Set Ratio divisor**

MFMUL=expression **Set Ratio multiplier**
where -256.0000 < Ratio < 256.0000

After the appropriate **MF#** command is issued, or the **MS** command has been issued, a floating point ratio can further be applied by the firmware. Since the SmartMotor is an integer machine, that floating point ratio is accomplished by dividing one number by another.

MFR Calculate Mode Follow Ratio

MSR Calculate Mode Step Ratio

Once a numerator and denominator have been specified, and the appropriate hardware mode is selected, the motor can be put into ratio mode with the **MFR** or **MSR** commands (**MSR** for ratioing incoming step and direction signals). The following example sets up a 10.5:1 relationship:

```
MF4          'Read in full quadrature decode
MFMUL=2      '10.5:1=21:2
MFDIV=21
D=0          'be sure D is zero
MFR          'Invoke calculation
G            'Start
```

Once in a ratio mode the **V=#** and **D=#** commands will still work. They will invoke a phase shift of length **D** at a relative rate determined by **V**. For that reason, **D** must be zeroed out before issuing an **MFR** or **MSR** command or unexpected shifting could be taking place. In applications such as a Web Press, this ability to phase shift can be very useful.

MC Mode Cam

A cam is a basically round but irregular shape that rotates and causes a follower to move up and down in a profile determined by the shape of the cam's exterior.

Since the beginning of industrialization, cams have been used to create complex, reciprocating motion. Cams are most often carved out of steel and changing them, or having them invoke motion a great distance away are impractical. The SmartMotor provides an electronic alternative. Putting an encoder on the rotating part of a machine, sending the signals to a SmartMotor and programming the cam profile into the SmartMotor allows for the same complex, repeating motions to be accomplished without any of the typical mechanical limitations.

BASE=expression Base length

Part of defining a Cam relationship is specifying how many incoming encoder counts there are for one full cam rotation. Simply set **BASE** equal to this number.

SIZE=expression Number of Cam data entries

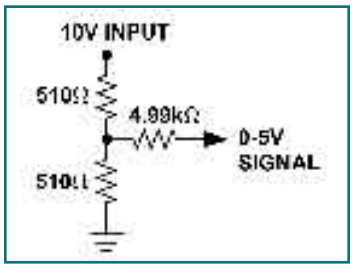
*Do not use variable **aa** through **zzz** while camming.*

The upper variable array space holds the cam profile data. To instruct the SmartMotor as to how many data points have been specified, set **SIZE** equal to that number. The cam firmware looks at words (16 bit numbers). The maximum number of words that can be used is 100. The cam firmware will perform linear interpolation between those entries, as well as between the last and the first as the cam progresses through the end of the table and back to the beginning. The cam table entries occupy the same space as variables **aa** through **yyy** which is the same space as the array variables. Invoking Cam Mode is done as follows:

```
BASE=2000          'Cam period
SIZE=25            'Data segments, this  defines the data
                      table size.
                   'CTR data, note the period at the end
aw[0] 0 10 20 30 40 50 60 70 80 90 100 110 120 120
   110 100 90 80 70 60 50 40 30 20 10 0.
MF0                'Reset external encoder to zero
O=0                'Reset internal encoder position
MC                 'Buffer CAM Mode
G                  'Start following the external encoder
                      using cam data
```

MD50 Drive Mode

The **MD50** command causes the SmartMotor to emulate a traditional servo and amplifier. In this mode, it can be used with yet another controller sending a standard +/-10Volt analog command signal. A small voltage divider is necessary to convert the +/-10 volts into the 0 to 5 volt signal the motor takes as input. The circuit considers the SmartMotor's input impedance. An additional device may be desired to take the single ended encoder signals coming out of the SmartMotor and make them differential for more noise immunity during their travel back to the external controller. An additional measure of optically isolating the encoder signals should be taken to avoid ground loops back to the control.



Voltage divider converting the +/-10 volt signal into a 0-5 volt signal

```
MD50               'Set Drive Mode
```

ENC0, ENC1 Encoder Select

The **ENC1** command causes the SmartMotor to servo off of an external encoder connected to inputs A and B. This can be useful if the external encoder has the advantage of being more accurate. The **ENC0** command restores the default mode of servoing off of the internal encoder.

```
ENC1            'Servo off of external encoder
ENC0            'Servo from internal encoder (default)
```

The following binary values can be tested by **IF** and **WHILE** control flow expressions, or assigned to any variable. They may all be reported using **RB**{bit} commands. Some may be reset using **Z**{bit} commands and some are reset when accessed. The first 8 states are reported in combination by the RS command. RW reports sixteen of these flags in combination.

By writing programs to periodically test these bits, a SmartMotor application can be very “smart” about its own inner-workings and doings.

Bo	Motor off	status bit 7
Bh	Excessive temperature	status bit 6
Be	Excessive position error	status bit 5
Bw	Wraparound occurred	status bit 4
Bi	Index report available	status bit 3
Bm	Real time negative limit	status bit 2
Bp	Real time positive limit	status bit 1
Bt	Trajectory in progress	status bit 0
Ba	Over current state occurred	
Bb	Parity error occurred	
Bc	Communication overflow occurred	
Bd	User math overflow occurred	
Bf	Communications framing error occurred	
Bk	Program check sum/EEPROM failure	
Bl	Historical left limit	
Br	Historical right limit	
Bs	Syntax error occurred	
Bu	User array index error occurred	
Bx	Hardware index input level	

If action is taken based on some of the error flags, the flag will need to be reset in order to look out for the next occurrence, or in some cases depending on how the code is written, in order to keep from acting over and over again on the same occurrence. The flags that need to be reset are listed. Their letter designator is preceded by the letter **Z** in the following table:

G also resets several system state flags

RESET SYSTEM STATE FLAGS

Za	Reset over current violation occurred
Zb	Reset parity error occurred
Zc	Reset com overflow error occurred
Zd	Reset user math overflow occurred
Zf	Reset communications framing error occurred
Zl	Reset historical left limit occurred
Zr	Reset historical right limit occurred
Zs	Reset syntax error occurred
Zu	Reset user array index error occurred
Zw	Reset wraparound occurred
ZS	Reset all **Z**{bit} state flags

The **TWAIT** command pauses program execution until motion is complete. Instead of using **TWAIT**, a routine could be written that does much more. To start with, the following code example would perform the same function as **TWAIT**:

```
WHILE Bt    'While trajectory
LOOP        'Loop back
```

Alternatively, the above routine could be augmented with code that took specific action in the event of an index signal as is shown in the following example

```
WHILE Bt    'While trajectory
  IF Bi           'Check index
    GOSUB500      'take care of it
  ENDIF           'end checking
LOOP        'Loop back
```

The following is a list of all of the commands used to relate to the SmartMotor's many I/O ports, grouped by port.

THE MAIN RS-232 PORT

ECHO — **ECHO** back all received characters

SADDR# — **S**et **ADDR**ess (0 to 120)

SILENT — Suppress print messages

TALK — Re-activate print message

SLEEP — Ignore all commands except **WAKE**

WAKE — Consider all following commands

BAUD19200 — Set baud rate to 19200 bps

OCHN (RS2,0,N,38400,1,8,D)
OpenChnl - RS-232, Channel 0, No parity, 38.4k bps, 1 stop, 8 data, as Data

OCHN (RS4,0,N,38400,1,8,C)
OpenChnl - RS-485 (w/adapter), Channel 0, No parity, 38.4k bps, 1 stop, 8 data, as Control

IF LEN>0 — Check to see if any (or how much) data is in the 16 byte input buffer, Data mode

c=GETCHR — Get byte from buffer into variable **c** for Data mode

PRINT (“Char Rcd:”,c,#13)
Print text, data and ASCII code for carriage return

THE G PORT

UGI — Redefine as general input

UGO — Redefine as general output (Open collector, pulled to 5V)

UG — Return pin to default start function, when low motor starts motion

UG=0 — Set G port Low (**UG=a** to set to variable **a**)

UG=1 — Set G port High (Open collector, weakly pulled to 5V internally)

a=UGI — Set variable **a** to digital input

a=UGA — Set **a** to analog input, 0 to 1023 = 0 to 5V

THE LIMIT PORTS C AND D

UCI	Redefine Right Limit as general input (**UDI** for Left Limit)
UCO	Redefine Right Limit as general output (**UDO** for Left Limit)
UCP	Return pin to limit function (**UDM** for Left Limit)
UC=0	Set Right Limit Low (**UD=0** for Left, or **UD=a** to set to variable **a**)
UC=1	Set Right Limit High **(UD=1** for Left Limit)
a=UCI	Set variable **a** to digital input (**UDI** for Left Limit)
a=UCA	Set **a** to analog input, 0 to 1023 = 0 to 5V (**UDA** for Left Limit)

COUNTER FUNCTIONS OF PORTS A AND B

MF4	Set **M**ode **F**ollow with full quadrature
MFR	Set **M**ode **F**ollow with ratio for gearing
MS	**M**ode **S**tep and Direction
MC	**M**ode **C**am
MF0	Set follow mode to zero and increment counter only
MS0	Set step mode to zero and increment counter only
a=CTR	Set variable **a** to counter value

GENERAL I/O FUNCTIONS OF PORTS A AND B

UAI	Set port A to input (**UBI** for port B)
UAO	Set port A to output (**UBO** for port B)
UA=0	Set port A Low (**UB=0** for port B, or **UB=a** to set to variable **a**)
UA=1	Set port A High (**UB=1** for port B)
a=UAI	Set variable **a** to digital input (**UBI** for port B)
a=UAA	Set **a** to analog input, 0 to 1023 = 0 to 5V (**UBA** for port B)

THE ANILINK PORT (USING I²C PROTOCOL)

AOUTB,c Send variable **c** out to Analog I/O board addressed as **B**

DOUTB0,c Send variable **c** out to Digital I/O board addressed as **B0**

c=AINB2 Set variable **c** to input 2 from Analog I/O board addressed as **B**

c=DINB0 Set variable **c** to input from Digital I/O board addressed as **B0**

PRINTB("Temp:",c,#32)
Print to LCD on network - text, data and ASCII code

THE ANILINK PORT (USING RS-485 PROTOCOL)

OCHN(RS4,1,N,38400,1,8,D)
OpenChnl - RS-485, Channel 1, No parity, 38.4k bps, 1 stop, 8 data, as Data

IF LEN1>0 Check to see if data is in the 16 byte input buffer

c=GETCHR1 Get byte from buffer into variable **c**

PRINT1("Char Rcd:",c,#13) Print text, data and ASCII code

ECHO1 **ECHO** back all received characters

SILENT1 Suppress print messages

SLEEP1 Ignore all commands except **WAKE**

WAKE1 Consider all following commands

THE ANILINK PORT AS GENERAL I/O

UEI Set port E to input (**UFI** for port F)

UEO Set port E to output (**UFO** for port F)

UE=0 Set port E Low (**UF=0** for port F, or **UF=c** to set to variable **c**)

UE=1 Set port E High (**UF=1** for port F)

c=UEI Set variable **c** to digital input (**UFI** for port F)

c=UEA Set **c** to analog input, 0 to 1023 = 0 to 5V (**UFA** for port F)

Ports A through G have internal 5k Ohm resistive pull-ups.

The standard SmartMotor brings out 5 volt power and ground, as well as seven I/O points. Each one has multiple functions. They are **UA, UB, UC, UD, UE, UF** and **UG** and have the following functions:

UA
Digital Input, TTL, 0 to 5 volts
Digital Output, TTL, 0 to 5 volts
Analog Input, 10 bit, 0 to 1023
External Encoder **A** Input
Step and Direction, Step Input

UB
Digital Input, TTL, 0 to 5 volts
Digital Output, TTL, 0 to 5 volts
Analog Input, 10 bit, 0 to 1023
External Encoder **B** Input
Step and Direction, Direction Input

UC
Digital Input, TTL, 0 to 5 volts
Digital Output, TTL, 0 to 5 volts
Analog Input, 10 bit, 0 to 1023
Positive Limit Input

UD
Digital Input, TTL, 0 to 5 volts
Digital Output, TTL, 0 to 5 volts
Analog Input, 10 bit, 0 to 1023
Negative Limit Input

UE
Digital Input, TTL, 0 to 5 volts
Digital Output, TTL, 0 to 5 volts
Analog Input, 10 bit, 0 to 1023
AniLink Data I/O
AniLink RS-485 Signal **A**

UF
Digital Input, TTL, 0 to 5 volts
Digital Output, TTL, 0 to 5 volts
Analog Input, 10 bit, 0 to 1023
AniLink Clock Output
AniLink RS-485 Signal **B**

UG
Digital Input, TTL, 0 to 5 volts
Digital Output, TTL, 0 to 5 volts
Analog Input, 10 bit, 0 to 1023
Start Motion (or **GO**) Input

When used as general I/O, these commands will select the port function.

UAI	Assign pin A to input state
UAO	Assign pin A to output state
UA=0	Set **UA** to zero volts
UA=1	Set **UA** to 5 volts
a=UAI	Read **UA** digital value, 0 or 1
a=UAA	Read **UA** analog voltage
UBI	Assign pin B to input state
UBO	Assign pin B to output state
UB=0	Set **UB** to zero volts
UB=1	Set **UB** to 5 volts
a=UBI	Read **UB** digital value, 0 or 1
a=UBA	Read **UB** analog voltage
UCI	Assign pin C to input state
UCO	Assign pin C to output state
UCP	Re-assign pin C to +limit action
UC=0	Set **UC** to zero volts
UC=1	Set **UC** to 5 volts
a=UCI	Read **UC** digital value, 0 or 1
a=UCA	Read **UC** analog voltage
UDI	Assign pin D to input state
UDO	Assign pin D to output state
UDM	Re-assign pin D to -limit action
UD=0	Set **UD** to zero volts
UD=1	Set **UD** to 5 volts
a=UDI	Read **UD** digital value, 0 or 1
a=UDA	Read **UD** analog voltage
UEI	Assign pin E to input state
UEO	Assign pin E to output state
UE=0	Set **UE** to zero volts
UE=1	Set **UE** to 5 volts
a=UEI	Read **UE** digital value, 0 or 1
a=UEA	Read **UE** analog voltage
UFI	Assign pin F to input state
UFO	Assign pin F to output state
UF=0	Set **UF** to zero volts
UF=1	Set **UF** to 5 volts
a=UFI	Read **UF** digital value, 0 or 1
a=UFA	Read **UF** analog voltage

UGI	Assign pin G to input state
UGO	Assign pin G to output state
UG	Re-assign pin G to "**GO**"
UG=0	Set **UG** to zero volts
UG=1	Set **UG** to 5 volts
a=UGI	Read **UG** digital value, 0 or 1
a=UGA	Read **UG** analog voltage

The **UAA**, **UBA**, **UCA**, **UDA**, **UEA**, **UFA** and **UGA** variables reflect the analog voltages at the port pins regardless of how the pins are configured. The analog voltage of any pin can be read without effecting it's current mode of operation in any way. For example, a pin could be used as an output and then the analog input value could be read to see if it happened to be shorted, or RS-485* signal bias could be monitored at ports E and F.

**RS-485 is not available as standard on SMXXX5 SmartMotors*

The encoder and step counting capabilities of ports A and B are described in the section on External Encoder Modes. The serial data capabilities of ports E and F are described in the section on communications.

ANILINK I/O MODULES

In the event the on-board I/O is not enough, additional I/O can be connected via the AniLink port. A variety of Analog and Digital I/O cards are available, as well as peripheral devices like LCD and LED displays, push-wheel input devices, pendants and more. These products communicate with the SmartMotor through the AniLink port using I^2C protocol.

OUTPUT ASSIGNMENTS

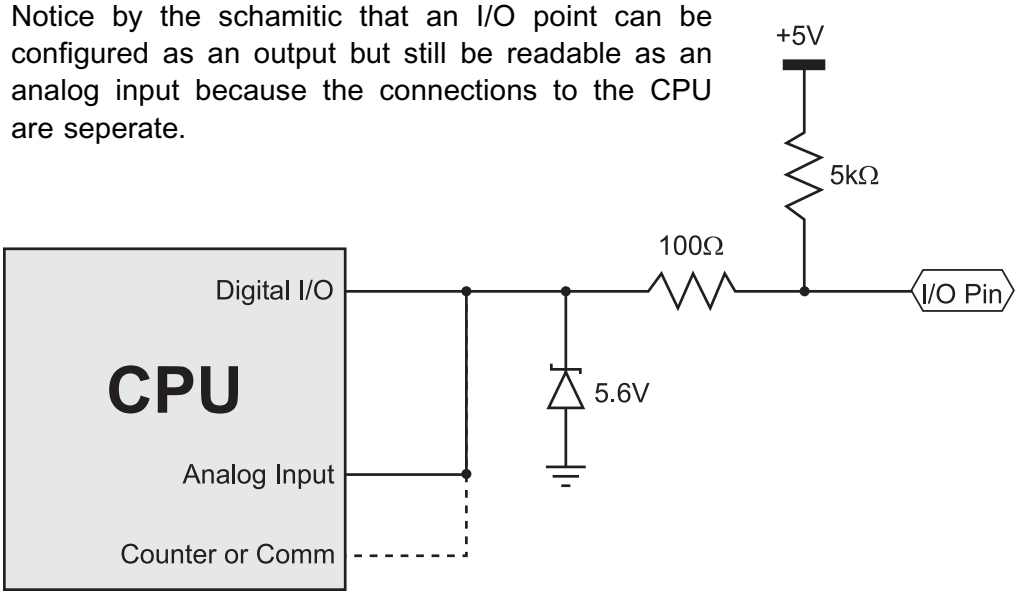
AOUT{address},exp	Output byte to analog address=A-H
DOUT{address}{ch},exp	Output byte to network, address=A-H, ch=0-63

INPUT ASSIGNMENTS

var=AIN{address}{input}	8 bit analog input from network, address=A-H, and input=1-4
var=DIN{address}{ch}	8 bit digital in from network, address=A-H, and ch=0-63

While all SmartMotor I/O is confined to operate between 0 and 5VDC, some circuitry exists to accomodate spikes above and below the operational range as long as those spikes are moderate and short lived.

Notice by the schamitic that an I/O point can be configured as an output but still be readable as an analog input because the connections to the CPU are seperate.



Knowing the SmartMotor's internal schematic can be useful when designing external interfaces.

All SmartMotor I/O points default to inputs when power is applied to the motor. It is the User Program that takes control after that. Because of the pull-up resistor, the voltage read at each port will be about 5VDC. When used as outputs to turn on external devices, it is highly recommended to design the system such that +5V is OFF and 0V is ON. This will prevent external equipment from being turned on immediately after power-up, before the User Program has a chance to take over.

Motor Connectors Pin Identification

4 Pin, 3 Phase Power

A Phase A
B Phase B
C Phase C
D Chassia

24 Pin Data I/O

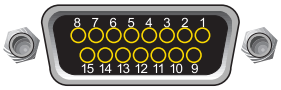
A RS-232 Ground
B RS-485 Ground
C Ground
D Ground
E RS-232 Transmit
F RS-232 Receive
G RS-485 A
H RS-485 B
J +5V Out, Fused
K Encoder A Out
L Encoder B Out
M Encoder I Out
N Ext Encoder A/Step/User I/O A
P Ext Encoder B/Direct/User I/O B
Q Reserved
R User I/O G
S Left Limit/User I/O D
T Right Limit/User I/O C
U AniLink Data/User I/O E
V AniLink Clock/User I/O F
W Reserved
X Reserved
Y Reserved
Z Reserved

7 Pin Combo D-Sub Power and I/O

A1 +20V to +48V DC
A2 Power Ground

1 Sync or I/O G
2 +5V Out
3 RS-232 Transmit
4 RS-232 Receive
5 RS-232 Ground

15 Pin D-Sub I/O



1 I/O A
2 I/O B
3 I/O C
4 I/O D
5 I/O E
6 I/O F
7 I/O G
8 Encoder A Out
9 Encoder B Out
10 SM RS-232 Transmit
11 SM RS-232 Receive
12 +5V Out
13 Ground
14 Power Ground
15 Power

Encoder I/O

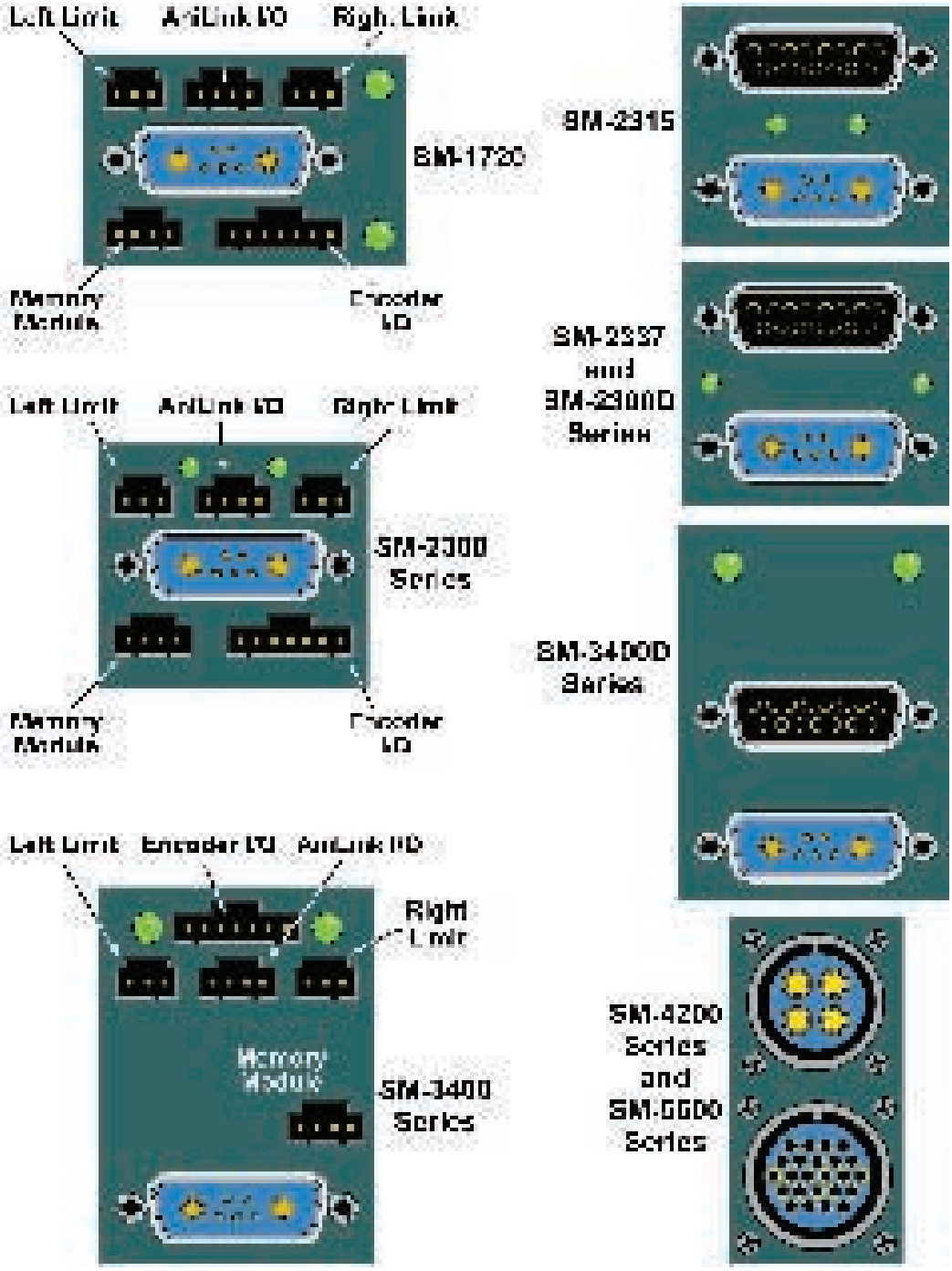
7 Ground
6 +5V Out
5 Encoder Index Out
4 Encoder B Out
3 Encoder A Out
2 Ext Encoder B/Step/User I/O B
1 Ext Encoder A/Direction/User I/O A

AniLink I/O

4 Clock/User I/O F
3 Data/User I/O E
2 Ground
1 +5V

Left/Right Limit I/O

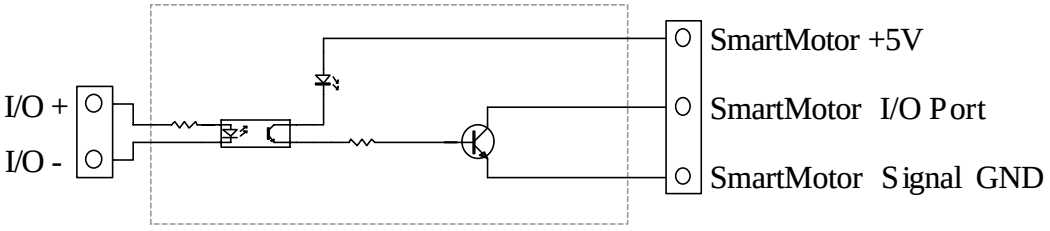
3 Limit Input
2 Ground
1 +5V Out



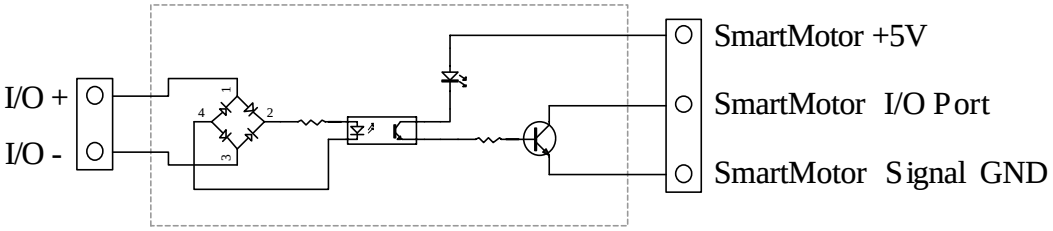
Motor Connector Locator

Please note the Memory Module location. This module uses the same type of connector as the AniLink I/O. If a Memory Module is plugged into the AniLink I/O, it won't break, but it won't work either.

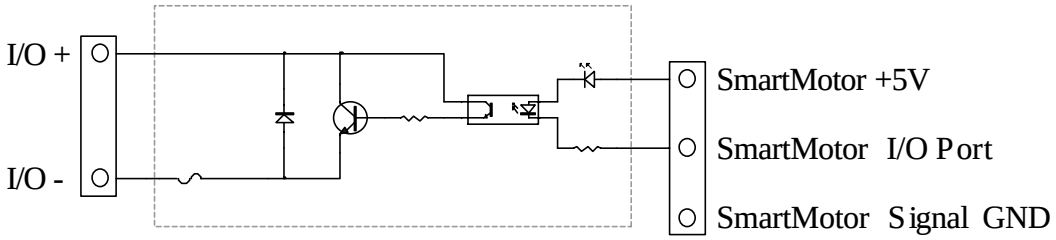
The 5 Volt Logic of the SmartMotor can interface to 24 Volt devices through the use of standard interface modules.



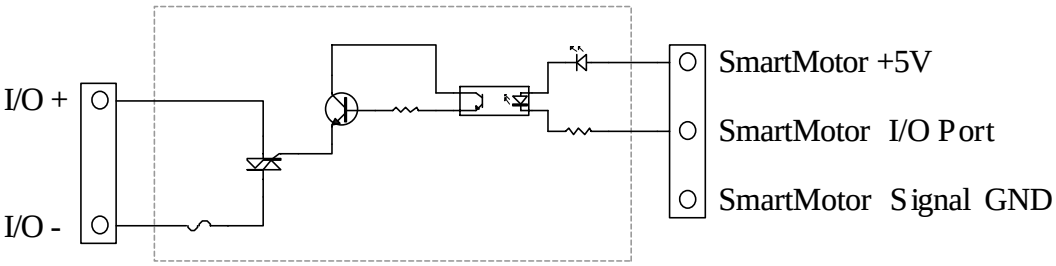
Typical DC Input Module (-IDC5 part numbers)



Typical AC Input Module (-IAC5 part numbers)



Typical DC Output Module (-ODC5 part numbers)



Typical AC Output Module (-OAC5 part numbers)

While there are a variety of options, the default mode for communicating with a SmartMotor is serial RS-232 for the main port. Each SmartMotor is equipped with a secondary serial port called81 the **AniLink** port. The **AniLink** port on a SmartMotor can be configured to communicate with either RS-485 or I^2C. The I^2C connects SmartMotor peripherals like LCD displays, I/O cards, etc., while the RS-485 will interface bar code readers, light curtains, and other "intelligent" peripherals including other SmartMotors if desired. SmartMotor models SMXXX5 do not have RS-485 capability in their **AniLink** ports.

When using I^2C, the SmartMotor is always the bus master. You cannot communicate between SmartMotors via I^2C.

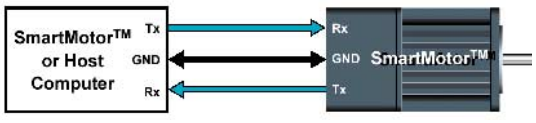
To maximize the flexibility of the SmartMotor, these serial communications ports are fully programmable with regard to bit-rate and protocol.

There is a sixteen-byte input buffer for the primary RS-232 port and another for the secondary RS-485 port. These buffers ensure that no arriving information is ever lost, although when either port is in data mode, it is the responsibility of the user program within the motor to keep up with the incoming data.

By default, the primary RS-232 channel, which shares a connector with the incoming power, is set up as a command port with the following default characteristics:

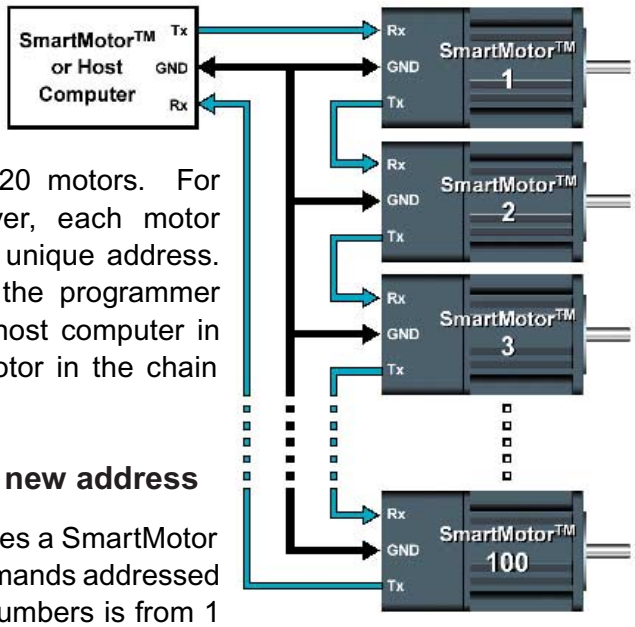
	Default:	**Other Options:**
Type:	RS-232	RS-485 (w/adapter)
Parity:	None	Odd or Even
Bit Rate:	9600	2400 to 38400
Stop Bits:	1	0 or 2
Data Bits:	8	7
Mode:	Command	Data
Echo:	Off	On

If the cable used is not provided by Animatics, make sure the SmartMotor's power and RS-232 connections are correct.



Because of the buffers on both sides there is no need for any hand shaking protocol when commanding the SmartMotor. Most commands execute in less time than it would take to receive the next one. Be careful to allow processes time to complete, particularly relatively slow processes like printing to a connected LCD display or executing a full subroutine. Since the **EEPROM** long term memory is slow to write, the terminal software does employ two way communication to regulate the download of a new program.

DAISY CHAINING RS-232



Multiple SmartMotors can be connected to a single RS-232 port as shown.

This diagram could be expanded to as many as 120 motors. For independent motion, however, each motor must be programmed with a unique address. In a multiple motor system the programmer has the choice of putting a host computer in control or having the first motor in the chain be in control of the rest.

SADDR# Set motor to new address

The **SADDR#** command causes a SmartMotor to respond exclusively to commands addressed to it. The range of address numbers is from 1 to 120. Once each motor in a chain has a unique address, each individual motor will communicate normally after its address is sent at least once over the chain. To send an address, add 128 to its value and output the binary result over the communication link. This puts the value above the ASCII character set, quickly and easily differentiating it from all other commands or data. The address needs to be sent only once until the host computer, or motor, wants to change it to something else. Sending out an address zero (128) will cause all motors to listen and is a great way to send global data such as a **G** for starting simultaneous motion in a chain. Once set, the address features work the same for RS-232 and RS-485 communications.

Unlike the RS-485 star topology, the consecutive nature of the RS-232 daisy-chain creates the opportunity for the chain to be independently addressed entirely from the host, rather than by having a uniquely addressed program in each motor. Setting up a system this way can add simplicity because the program in each motor can be exactly the same. If the **RUN?** Command is the first in each of the motor's programs, the programs will not start upon power up. Addressing can be worked out by the host prior to the programs being started later by the host sending the **RUN** command globally.

SLEEP, SLEEP1 Assert sleep mode

WAKE, WAKE1 De-assert SLEEP

Telling a motor to sleep causes it to ignore all commands except the **WAKE** command. This feature can often be useful, particularly when establishing unique addresses in a chain of motors. A **1** at the end of the command specifies the AniLink RS-485 port.

ECHO, ECHO1 **ECHO input**

ECHO_OFF, ECHO_OFF1 **De-assert ECHO**

The **ECHO** and **ECHO_OFF** commands toggle the echoing of data input. Because the motors do not echo character input by default, consecutive commands can be presented, configuring them with unique addresses, one at a time. If the host computer or controller sent out the following command sequence, each motor would have a unique and consecutive address.

If a daisy chain of SmartMotors have been powered off and back on, the following commands can be entered into the **SmartMotor Interface** to address the motors (0 equals 128, 1 equals 129, etc.). Some delay should be inserted between commands when sending them from a host computer.

```
0SADDR1
1ECHO
1SLEEP
0SADDR2
2ECHO
2SLEEP
0SADDR3
3ECHO
0WAKE
```

Commanded by a user program in the first motor, instead of a host, the same daisy chain could be addressed with the following sequence:

```
SADDR1                          'Address the first motor
ECHO                            'Echo for host data
PRINT(#128,"SADDR2",#13)        '0SADDR2
WAIT=10                         'Allow time
PRINT(#130,"ECHO",#13)          '2ECHO
WAIT=10
PRINT(#130,"SLEEP",#13)         '2SLEEP
WAIT=10
PRINT(#128,"SADDR3",#13)        '0SADDR3
WAIT=10
PRINT(#131,"ECHO",#13)          '3ECHO
WAIT=10
PRINT(#128,"WAKE",#13)          '0WAKE
WAIT=10
```

The two communications ports have enormous flexibility. To select from the vast array of options, use the **OCHN** command.

OCHN

	Options:	
Type:	RS2, RS4	RS-232 or RS-485
Channel:	0, 1 or 2	0=Main, 1=AniLink
Parity:	N, O or E	None, Odd or Even
Bit rate:	2400, 4800, 9600, 19200, 38400 baud	
Stop bits:	0, 1 or 2	
Data bits:	7 or 8	
Mode:	C or D	Command or Data

Here is an example of the **OCHN** command:

```
OCHN(RS2,0,N,38400,1,8,D)
```

If the primary communication channel (0) is opened as an RS-485 port, it will assume the RS-485 adapter is connected to it. If that is the case then pin **G** in the same connector is assigned the task of directing the adapter to be in Transmit or Receive mode in accordance with the motor's communication activity and will no longer be useful as an I/O port to the outside world.

CCHN(type,channel) Close a communications channel

Use the **CCHN** command to close a communications port when desired.

BAUD# Set BAUD rate of main port

The **BAUD#** command presents a convenient way of changing only the bit rate of the main channel. The number can be from 2400 to 38400 bps.

PRINT( ), PRINT1( ) Print to RS-232 or AniLink channel

A variety of data formats can exist within the parentheses of the **PRINT( )** command. A text string is marked as such by enclosing it between double quotation marks. Variables can be placed between the parentheses as well as two variables separated by one operator. To send out a specific byte value, prefix the value with the **#** sign and represent the value with as many as three decimal digits ranging from 0 to 255. Multiple types of data can be sent in a single **PRINT( )** statement by separating the entries with commas. Do not use spaces outside of text strings because SmartMotors use spaces as delimiters along with carriage returns and line feeds.

The following are all valid print statements and will transmit data through the main RS-232 channel:

```
PRINT("Hello World")   'text
PRINT(a*b)             'exp.
PRINT(#32)             'data
PRINT("A",a,a*b,#13)   'all
```

PRINT1 prints to the AniLink port with RS-485 protocol while **PRINTA** prints to the AniLink port using I^2C protocol in such a way as to send data to an LCD display or standard parallel input line printer (with a DIO-100 card on the AniLink bus).

SILENT, SILENT1 Suppress PRINT() outputs

TALK, TALK1 De-assert silent mode

The **SILENT** mode causes all **PRINT( )** output to be suppressed. This is useful when talking to a chain of motors from a host, when the chain would otherwise be talking within itself because of programs executing that contain **PRINT( )** commands.

! Wait for RS-232 character to be received

A single exclamation mark will cause program execution to stop until a character is received. This can be handy under certain circumstances like debugging a program in real time.

a=CHN0, a=CHN1 RS-485 communications error flags

The **CHN0** and **CHN1** variables hold binary coded information about the historical errors experienced by the two communications channels. The information is as follows:

Bit	Value	Meaning
0	1	Buffer overflow
1	2	Framing error
2	4	Command scan error
3	8	Parity error

A subroutine that printed the errors to an LCD display would look like the following:

```
C911
   IF CHN0          'If CHN0 != 0
      DOUT0,1       'Home LCD cursor
      IF CHN0&1
         PRINTA("BUFFER OVERFLOW")
      ENDIF
      IF CHN0&2
         PRINTA("FRAMING ERROR")
      ENDIF
      IF CHN0&4
         PRINTA("COMMAND SCAN ERROR")
      ENDIF
      IF CHN0&8
         PRINTA("PARITY ERROR")
```

```
        ENDIF
        CHN0=0          'Reset CHN0
      ENDIF
    RETURN
```

a=ADDR Motor's self address

If the motor's address (**ADDR**) is set by an external source, it may still be useful for the program in the motor to know what address it is set to. When a motor is set to an address, the **ADDR** variable will reflect that address from 1 to 120.

GETTING DATA FROM A COM PORT

If a com port is in Command Mode, then the motor will simply respond to arriving commands it recognizes. If the port is opened in Data Mode, however, then incoming data will start to fill the 16 byte buffer until it is retrieved with the **GETCHR** command.

a=LEN	Number of characters in RS-232 buffer
a=LEN1	Number of characters in RS-485 buffer
a=GETCHR	Get character from RS-232 buffer
a=GETCHR1	Get character from RS-485 buffer

The buffer is a standard **FIFO** (**F**irst **I**n **F**irst **O**ut) buffer. This means that if the letter **A** is the first character the buffer receives, then it will be the first byte offered to the **GETCHR** command. The buffer exists to make sure that no data is lost, even if the program is not retrieving the data at just the right time. Two things are very important when dealing with a data buffer for the protection of the data:

1) Never **GETCHR** if there is no **CHR** to **GET**.
2) Never let the buffer overflow.

The **LEN** variable holds the number of characters in the buffer. A program must see that the **LEN** is greater than zero before issuing a command like: **a=GETCHR**. Likewise, it's necessary to arrange the application so that, overall, data will be pulled out of the buffer faster than it comes in.

The ability to configure the communication ports for any protocol as well as to both transmit and receive data allows the SmartMotor to interface to a vast array of RS-232 and RS-485 devices. Some of the typical devices that would interface with SmartMotors over the communication interface are:

1) Other SmartMotors
2) Bar Code Readers
3) Light Curtains
4) Terminals
5) Printers

The following is an example program that repeatedly transmits a message to an external device (in this case another SmartMotor) and then takes a number back from the device as a series of ASCII letter digits, each ranging from 0 to 9. A carriage return character will mark the end of the received data. The program will use that data as a position to move to.

```
A=500              'Preset Accel.
V=1000000          'Preset Vel.
P=0                'Zero out Pos.
O=0                'Declare origin
G                  'Servo in place
OCHN(RS2,0,N,9600,1,8,D)
PRINT("RP",#13)
C0
   IF LEN          'Check for chars
      a=GETCHR     'Get char
      IF a==13     'If carriage return
         G         'Start motion
         P=0       'Reset buffered P to zero
         PRINT("RP",#13)     'Next
      ELSE
         P=P*10    'Shift buffered P
         a=a-48    'Adjust for ASCII
         P=P+a     'Build buffered P
      ENDIF
   ENDIF
GOTO0              'Loop forever
```

The ASCII code for zero is 48. The other nine digits count up from there so the ASCII code can be converted to a useful number by subtracting the value of 0 (ASCII 48). The example assumes that the most significant digits will be returned first. Any time it sees a new digit, it multiplies the previous quantity by 10 to shift it over and then adds the new digit as the least significant. Once a carriage return is seen (ASCII 13), motion starts. After motion is started, **P** (Position) is reset to zero in preparation for building up again. **P** is buffered so it will not do anything until the **G** command is issued.

PID FILTER CONTROL

The SmartMotor™ includes a very high quality, high performance brushless D.C. servomotor. It has a rotor with extremely powerful rare earth magnets and a stator (the outside, stationary part) that is a densely wound multi-slotted electro-magnet.

Controlling the position of a brushless D.C. servo's rotor with only electromagnetism working as a lever is like pulling a sled with a rubber band. Accurate control would seem impossible.

The parameters that makes it all work are found in the **PID** (**P**roportional, **I**ntegral, **D**erivative) filter section. These are the three fundamental coefficients to a mathematical algorithm that intelligently recalculates and delivers the power needed by the motor about 4,000 times per second. The input to the **PID** filter is the instantaneous actual position minus the desired position, be it at rest, or part of an ongoing trajectory. This difference is called the error.

While the Derivative term usually acts to dampen instability, this is not the true definition of the term. It is possible to cause instability by setting the Derivative term too high.

The **P**roportional parameter of the filter creates a simple spring constant. The further the shaft is rotated away from its target position, the more power is delivered to return it. With this as the only parameter the motor shaft would respond just as the end of a spring would if it was grabbed and twisted.

If the spring is twisted and let go it will vibrate wildly. This sort of vibration is hazardous to most mechanisms. In this scenario a shock absorber is added to cancel the vibrations which is the equivalent of what the **D**erivative parameter does. If a person sat on the fender of a car, it would dip down because of the additional weight based on the constant of the car's spring. It would not be known if the shocks were good or bad. If the bumper was jumped up and down on, however, it would quickly become apparent whether the shock absorbers were working or not. That's because they are not activated by position but rather by speed. The **D**erivative parameter steals power away as a function of the rate of change of the overall filter output. The parameter gets its name from the fact that the derivative of position is speed. Electronically stealing power based on the magnitude of the motor shafts vibration has the same effect as putting a shock absorber in the system, and the algorithm never goes bad.

Even with the two parameters a situation can arise that will cause the servo to leave its target created by "dead weight". If a constant torque is applied to the end of the shaft, the shaft will comply until the deflection causes the **P**roportional parameter to rise to the equivalent torque. There is no speed so the **D**erivative parameter has no effect. As long as the torque is there, the motor's shaft will be off of its target.

That's where the **I**ntegral parameter comes in. The **I**ntegral parameter mounts an opposing force that is a function of time. As time passes and there is a deflection present, the **I**ntegral parameter will add a little force to bring it back on target with each **PID** cycle. There is also a separate parameter (**KL**) used to limit the **I**ntegral parameter's scope of what it

can do so as not to over react.

Each of these parameters have their own scaling factor to tailor the overall performance of the filter to the specific load conditions of any one particular application. The scaling factors are as follows:

KP	**P**roportional
KI	**I**ntegral
KD	**D**erivative
KL	Integral Limit

TUNING THE FILTER

The task of tuning the filter is complicated by the fact that the parameters are so interdependent. A change in one can shift the optimal settings of the others. The automatic utility makes all of the settings easy, but it still may be necessary to know how to tune a servo.

When tuning the motor it is useful to have the status monitor running which will monitor various bits of information that will reflect the motors performance.

KP=exp	Set **KP**, proportional coefficient
KI=exp	Set **KI**, time-error coefficient
KD=exp	Set **KD**, damping coefficient
KL=exp	Set **KL**, time-error term limit
F	Update **PID** filter

The main objective in tuning a servo is to get **KP** as high as possible, while maintaining stability. The higher the **KP** the stiffer the system and the more under control it is. A good start is to simply query what to begin with (**RKP**) and then start increasing it 10% to 20% at a time. It is a good idea to start with **KI** equal to zero. Keep in mind that the new settings do not take effect until the **F** command is issued. Each time **KP** is raised, try physically to destabilize the system, by bumping or twisting it. Or, have a program loop cycling that invokes abrupt motions. As long as the motor always settles to a quiet rest, keep raising **KP**. Of course if the **SMI Tuning Utility** is being used, it will employ a step function and show more precisely what the reaction is.

As soon as the limit is reached, find the appropriate derivative compensation. Move **KD** up and down until the position is found that gives the quickest stability. If **KD** is way too high, there will be a grinding sound. It is not really grinding, but it is a sign to go the other way. A good tune is not only stable, but reasonably quiet. After optimizing **KD**, it may be possible to raise **KP** a little more. Keep going back and forth until there's nothing left to improve the stiffness of the system. After that it's time to take a look at **KI**.

KI, in most cases, is used to compensate for friction. Without it the SmartMotor will never exactly reach the target. Begin with **KI** equal to zero and **KL** equal to 1000. Move the motor off target and start increasing **KI** and **KL**. Keep **KL** at least ten times **KI** during this phase.

Continue to increase **KI** until the motor always reaches its target, and once that happens add about 30% to **KI** and start bringing down **KL** until it hampers the ability for the **KI** term to close the position precisely to target. Once that point is reached, increase **KL** by about 30% as well. The **I**ntegral term needs to be strong enough to overcome friction, but the limit needs to be set so that an unruly amount of power will not be delivered if the mechanism were to jam or simply find itself against one of its ends of travel.

E=expression Set maximum position error

The difference between where the motor shaft is and where it is supposed to be is appropriately called the “error”. The magnitude and sign of the error is delivered to the motor in the form of torque, after it is put through the **PID** filter. The higher the error, the more out of control the motor is. Therefore, it is often useful to put a limit on the allowable error, after which time the motor will be turned off. That is what the **E** command is for. It defaults to 1000 encoder counts, but can be set from 1 to 32,000.

There are still more parameters that can be utilized to reduce the position error of a dynamic application. Most of the forces that aggravate a **PID** loop through the execution of a motion trajectory are unpredictable, but there are some that can be predicted and further eliminated preemptively.

KG=expression Set KG, Gravity offset term

The simplest of these is gravity. Why burden the **PID** loop with the effects of gravity in a vertical load application, if it can simply be weeded out. If in a particular application, motion would occur with the power off due to gravity, a constant offset can be incorporated into the filter to balance the system. **KG** is the term. **KG** can range from -8388608 to 8388607. To tune **KG,** simply make changes to **KG** until the load equally favors upward and downward motion.

KV=expression Set KVff, velocity feed forward

Another predictable cause of position error is the natural latency of the **PID** loop itself. At higher speeds, because the calculation takes a finite amount of time, the result is somewhat “old news”. The higher the speed, the more the actual motor position will slightly lag the trajectory calculated position. This can be programmed out with the **KV** term. KV can range from zero to 65,535. Typical values range in the low hundreds. To tune **KV** simply run the motor at a constant speed, if the application will allow, and increase **KV** until the error gets reduced to near zero and stays there. The error can be seen in real time by activating the **Monitor Status** window in the **SMI** program.

KA=expression **Set KAff, acceleration feed forward**

Force equals mass times acceleration. If the SmartMotor is accelerating a mass, it will be exerting a force during that acceleration. This force will disappear immediately upon reaching the cruising speed. This momentary torque during acceleration is also predictable and need not aggravate the **PID** filter. It's effects can be programmed out with the **KA** term. It is a little more difficult to tune **KA**, especially with hardware attached. The objective is to arrive at a value that will close the position error during the acceleration and deceleration phases. It is better to tune **KA** with **KI** set to zero because **KI** will address this constant force in another way. It is best to have **KA** address 100% of the forces due to acceleration, and leave the **KI** term to adjust for friction.

KS=expression **Set KS, dampening sample rate**

Reduce the sampling rate of the derivative term, **KD**, with the **KS** term. This can sometimes add stability to very high inertial loads. Useful values of **KS** range from 1 (the default) to 20. Results will vary from application to application.

The **PID** rate of the SmartMotor can be slowed down.

PID1 Set normal **PID** update rate

PID2 Divide normal **PID** update rate by 2

PID4 Divide normal **PID** update rate by 4

PID8 Divide normal **PID** update rate by 8

The trajectory and **PID** filter calculations occur within the SmartMotor™ 4069 times per second. That is faster than is necessary for very good control, especially with the larger motors. A reduction in the **PID** rate can result in an increase in the SmartMotor™ application program execution rate. The **PID2** command will divide the **PID** rate by two, and the others even more. The most dramatic effect on program execution rate occurs with **PID4**. **PID8** does little more and is encroaching upon poor control. If the **PID** rate is lowered, keep in mind that this is the "sample" rate that is the basis for **V**elocity values, **A**cceleration values, **PID** coefficients and **WAIT** times. If the rate is cut in half, expect to do the following to keep all else the same:

Halve **WAIT** times

Double **V**elocity

Increase **A**cceleration by a factor of 104

KGON **Change Drive Characteristic for Vertical Application**

KGOFF **Restore Drive Characteristic to Default**

Vertical applications can be particularly hard to tune, even with the **KG** term. Often this is seen in an awkward sound or vibration occurring when the motor decelerates on its way down. The filter and the drive electronics are challenged to deal with the situation where the motor wants to go in the

same direction it is told to go, entirely on its own volition. The SmartMotor is equipped with a special drive mode designed to deal with this very situation. This mode of operation is invoked with the **KGON** command. Because this mode is so different from the standard drive mode, it will be necessary to tune the motor once again. The reason this more stable drive mode is not the default is because like most good things, it comes at a cost. The SmartMotor's amplifier is not as efficient at converting current to torque in this mode as it is in the default mode and so it is necessary to verify the motor's reasonably cool operation when **KGON** is in use. Use **KGOFF** to revert back to the standard drive mode.

CURRENT LIMIT CONTROL

AMPS=expression Set current limit, 0 to 1023

In some applications, if the motor misapplied full power, the attached mechanism could be damaged. It can be useful to reduce the maximum amount of current available thus limiting the torque the motor can put out. Use the **AMPS** command with a number, variable or expression within the range of 0 to 1023. The units are tenths of a percent of full scale peak current, and varies in actual torque with the size of the SmartMotor.

APPENDIX A: UNDERSTANDING BINARY DATA

The SmartMotor's[tm] language allows the programmer to access data on the binary level. Understanding binary data is very easy and useful when programming the SmartMotor or any electronic device. What follows is an explanation of how binary data works.

All digital computer data is stored as binary information. A binary element is one that has only two states, commonly described as "on" and "off" or "one" and "zero". A light switch is a binary element. It can either be "on" or "off". A computer's memory is nothing but a vast array of binary switches called "bits".

The power of a computer comes from the speed and sophistication with which it manipulates these bits to accomplish higher tasks. The first step towards these higher goals is to organize these bits in such a way that they can describe things more complicated than "off" or "on".

Different numbers of bits are used to make up different building blocks of data. They are most commonly described as follows:

Four bits	=	Nibble
Eight bits	=	Byte
Sixteen bits	=	Word
Thirty two bits	=	Long

One bit has two possible states, on or off. Every time a bit is added, the possible number of states is doubled. Two bits have four possible states. They are as follows:

00	off-off
01	off-on
10	on-off
11	on-on

A nibble has 16 possible states. A byte has 256 and a Long has billions of possible combinations.

Because a byte of information has 256 possible states, it can reflect a number from zero to 255. This is elegantly done by assigning each bit a value of twice the one before it, starting with one. Each bit value becomes as follows:

Bit	Value
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

If all their values are added together the result is 255. By leaving particular bits out any sum between zero and 255 can be created. Look at the following example bytes and their decimal values:

Byte	Value
0 0 0 0 0 0 0 0	0
0 0 0 0 0 0 0 1	1
0 0 0 0 0 0 1 0	2
0 0 0 0 0 0 1 1	3
0 0 0 1 0 0 0 0	16
1 0 0 0 0 0 0 0	128
1 0 0 0 0 0 0 1	129
1 1 1 1 1 1 1 1	255

Consider the following two bytes of information:

Byte	Value
0 0 1 1 1 1 0 0	60
0 0 0 1 1 1 1 0	30

To make use of the limited memory available with micro controllers that can fit into a SmartMotor, there are occasions where every bit is used. One example is the status byte. A single value can be uploaded from a SmartMotor and have coded into it, in binary, eight or sixteen independent bits of information.

The following is the status byte and its coded information:

Name	Description	Bit	Value
Bo	Motor OFF	7	128
Bh	Excessive temp.	6	64
Be	Excessive pos. err.	5	32
Bw	Wraparound	4	16
Bi	Index reportable	3	8
Bm	Real time neg. lim.	2	4
Bp	Real time pos. lim.	1	2
Bt	Trajectory going	0	1

There are two useful mathematical operators that work on binary data, the "**&**" (and) and the "**|**" (or). The "**&**" compares two bytes, words or longs and looks for what they have in common. The resulting data has ones only where there were ones in both the first byte and the second. The "**|**" looks for a one in the same location of either the first data field or the second. Both functions are illustrated in the following example:

A	B	A&B	A|B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Knowing how the binary data works will enable shorter and faster code to be written. The following are two code examples that are looking to see if both limit inputs are high. One does this without taking advantage of the binary operator while the second shows how using the binary operator makes the code shorter, and therefor faster.

Example 1:

```
IF Bm            'look for - lim high
   IF Bp         'loof for + lim high
      GOSUB100   'handle it
   ENDIF
ENDIF
```

Example 2:

```
IF S&6           'look at both lim
   GOSUB100      'handle it
ENDIF
```

Both examples will execute subroutine 100 if both limit inputs are high. By "anding" the status byte (S) by six, the second routine filters out all of the other status information. If either limit is high, then the result will be non-zero and subroutine 100 will execute. Example two uses much less code than example one and will run much faster as a part of a larger program loop.

The next two examples show how the use of the "|" operator can improve program size and execution speed:

Example 3:

```
IF UAI           'look for input A
   GOSUB200      'handle it
ENDIF
IF UBI           'look for input B
   GOSUB200      'handle it
ENDIF
```

Example 4:

```
IF UAI|UBI       'look at both A,B
   GOSUB200      'handle it
ENDIF
```

Both examples 3 and 4 accomplish the same task with different levels of efficiency.

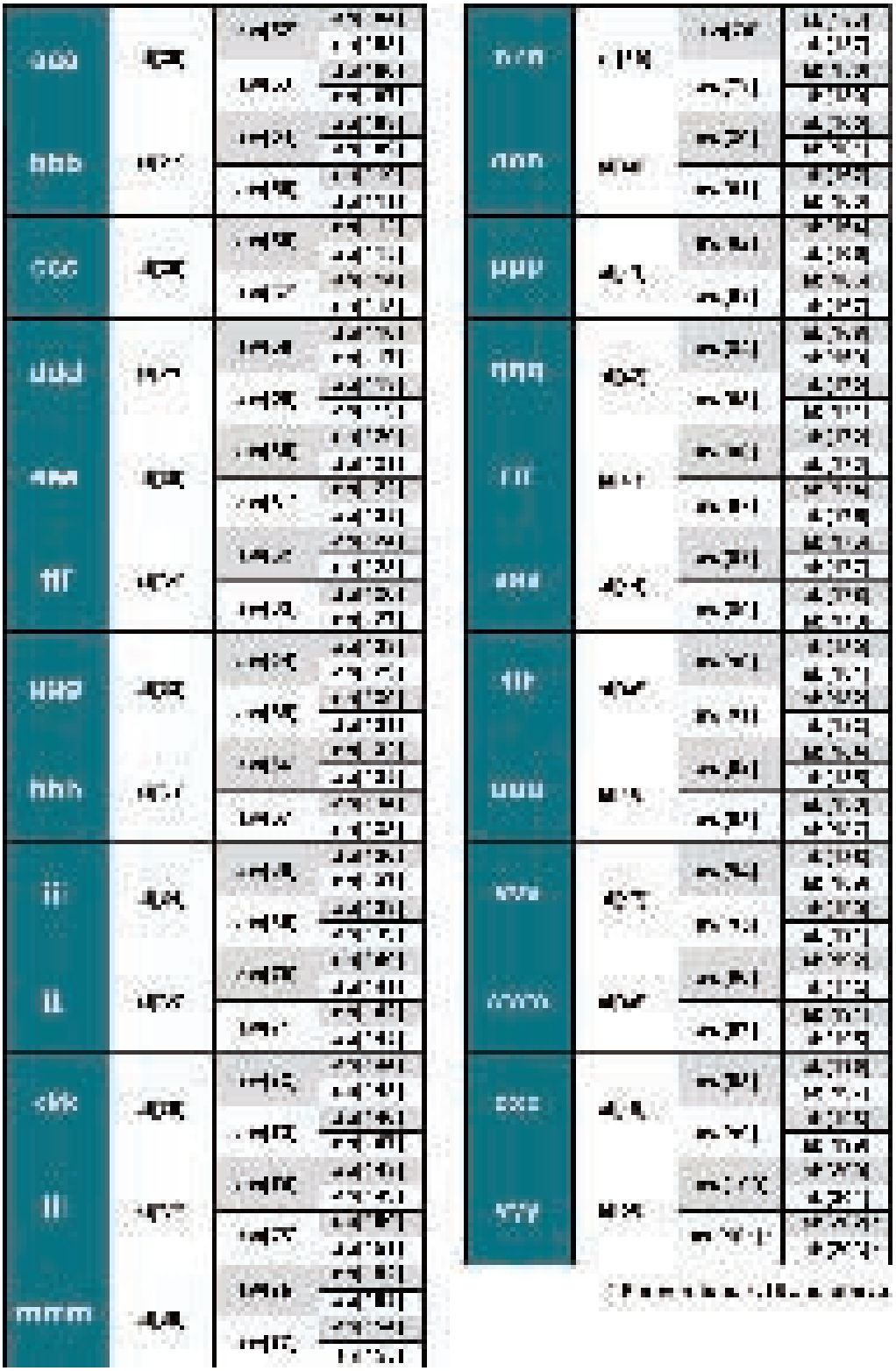
ASCIIIS an acronym for American Standard Code for Information Interchange. It refers to the convention established to relate characters, symbols and functions to binary data. If a SmartMotor is asked its position over the RS-232 link, and it is at position 1, it will not return a byte of value one, but instead will return the ASCII code for 1 which is binary value 49. That is why it appears on a terminal screen as the numeral 1.

The ASCII character set is as follows:

0	NUL	32	SP	64	@	96	’
1	SOH	33	!	65	A	97	a
2	STX	34	“	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	‘	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FC	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	Del

APPENDIX C: USER ASSIGNED VARIABLES MEMORY MAP

APPENDIX C: USER ASSIGNED VARIABLES MEMORY MAP



! (exclamation point)	
(space)	Single space between user variables
@P	Current position
@PE	Current position error
@V	Current velocity
a . . . z	User variables
aa . . . zzz	More user variables
al[index]	Array variable 32 bit
aw[index]	Array variable 16 bit
ab[index]	Array variable 8 bit
A=exp	Set acceleration
ADDR	Motor's self address variable
AIN{port}{channel}	Assign input byte from module
AMPS=expression	Set PWM drive signal limit
AOUT{port}{expression}	Output analog byte to module
Ba	Over current status bit
Bb	Parity error status bit
Bc	Communication overflow status bit
Bd	Math overflow status bit
Be	Excessive position error status bit
Bf	Communications framing error status bit
Bh	Excessive temperature status bit
Bi	Index captured status bit
Bk	EEPROM data integrity status bit
Bl	Historical left limit status bit
Bm	Real time left limit status bit
Bo	Motor off status bit
Bp	Real time right limit status bit
Br	Historical right limit status bit

Bs	Syntax error status bit
Bt	Trajectory in progress status bit
Bu	Array index error status bit
Bv	EEPROM locked state (obsolete)
Bw	Encoder wrap around status bit
Bx	Real time index inptut status bit
BASE	Cam encoder count cycle length
BAUD	Host communications control
BRKENG	Brake engage
BRKRLS	Brake release
BRKSRV	Brake without servo
BRKTRJ	Brake without trajectory
BREAK	Program execution flow control
C#	Program subroutine label
CCHN{type}{channel}	Close communications channel
CHN0	RS-232 communications error flags
CHN1	RS-485 communications error flags
CLK	Hardware clock variable
CTR	Second encoder/step and direction counter
D=exp	Set relative distance
DEFAULT	Switch-case structure element
DIN{port}{channel}	Input byte from module
DOUT{port}{channel}{expression}	Output byte to module
E=expression	Set allowable position error
ECHO	Echo input data back out main channel
ECHO_OFF	Stop echo main channel
ECHO1	Echo input data back out second channel
ECHO1_OFF	Stop echo second channel
ELSE	If structure element
ENC0	Select internal encoder for servo

ENC1	Select external encoder for servo
END	End program
ENDIF	End **IF** statement
EPTR=expression	Set data EEPROM pointer
ES400	Slow data EEPROM read/write speed
ES1000	Increase data EEPROM read/write speed
F	Load filter
F=expression	Special functions control
G	Start motion (GO)
GETCHR	Get character from main comm channel
GETCHR1	Get character from second comm channel
GOSUB#	Call a subroutine
GOTO#	Branch program execution to a label
I (capital i)	Hardware index position variable
IF expression	Conditional test
KA=expression	PID acceleration feed-forward
KD=expression	PID derivative compensation
KG=expression	PID gravity compensation
KGOFF	PID gravity mode off
KGON	PID gravity mode on
KI=expression	PID integral compensation
KL=expression	PID integral limit
KP=expression	PID proportional compensation
KS=expression	PID derivative term sample rate
KV=expression	PID velocity feed forward
LEN	Main comm chnl buffer fill level, data mode
LEN1	Second comm chnl buffer fill level, data mode
LIMD	Enable directional constraints on limit inputs
LIMH	Limit active high
LIML	Limit active low

LIMN	Restore non-directional limits
LOAD	Initiate program download to motor
LOOP	While structure element
MC	Enable cam mode
MC2	Enable cam mode with position scaled x2
MC4	Enable cam mode with position scaled x4
MC8	Enable cam mode with position scaled x8
MD	Enable contouring mode
MD50	Enable drive mode
MF0	Set mode follow for variable only
MF1	Configure follow hardware for x1 scaling
MF2	Configure follow hardware for x2 scaling
MF4	Configure follow hardware for x4 scaling
MFDIV	Mode follow with ratio divisor
MFMUL	Mode follow with ratio multiplier
MFR	Initiate mode follow ratio calculation
MP	Enable position mode
MS	Enable step and direction input mode
MS0	Configure step and direction for variable only
MSR	Initiate mode step ratio calculation
MT	Enable torque mode
MV	Enable velocity mode
O=expression	Set origin
OCHN	Open main communications channel
OFF	Stop servoing the motor
P=expression	Set position
PID1	Restore PID sample rate to default
PID2	Divide PID sample rate by two
PID4	Divide PID sample rate by four
PID8	Divide PID sample rate by eight

PRINT{expression}	Print data to main comm channel
PRINT1{expression}	Print data to second comm channel
PRINT{port}{expression}	Print data to AniLink peripheral
Q	Report status in contouring mode
Ra . . . Rz	Report variables
Raa . . . Rzz	Report variables
Raaa . . . Rzzz	Report variables
Rab[index]	Report byte array variables (8-bit)
Ral[index]	Report long array variables (32-bit)
Raw[index]	Report word array variables (16-bit)
RA	Report acceleration
RAIN{expression}{input}	Report value from analog AniLink card
RAMPS	Report assigned max. drive PWM limit
RBa	Report over current status
RBb	Report parity error status
RBc	Report communications error status
RBd	Report user math overflow status
RBe	Report position error status
RBf	Report communications framing error status
RBh	Report overheat status
RBi	Report index status
RBk	Report EEPROM read/write status
RBl	Report historical left limit status
RBm	Report negative limit status
RBo	Report motor off status
RBp	Report positive limit status
RBr	Report historical right limit status
RBs	Report program scan status
RBt	Report trajectory status
RBu	Report user array index status

RBw	Report wrap around status
RBx	Report hardware indexinput level
RCHN	Report combined communications status
RCHN0	Report RS-232 communications status
RCHN1	Report RS-485 communications status
RCS	Report RS-232 communications check sum
RCS1	Report RS-485 communications check sum
RCTR	Report secondary counter
RD	Return buffered move distance value
RDIN{port}{channel}	Report value from digital AniLink card
RE	Report buffered maximum position error
RES=expression	High resolution encoder control
RETURN	Return from subroutine
RI	Report last stored index position
RKA	Report buffered acceleration feed forward coef.
RKD	Report buffered derivative coefficient
RKG	Report buffered gravity coefficient
RKI	Report buffered integral coefficient
RKL	Report buffered integral limit
RKP	Report buffered proportional coefficient
RKS	Report buffered sampling interval
RKV	Report buffered velocity feed forward coefficient
RMODE	Report current mode of operation
RP	Report present position
RPE	Report present position error
RPW	Report position and status
RS	Report status byte
RT	Report current requested torque
RUN	Execute stored program
RUN?	Override automatic program execution

RV	Report velocity
RW	Report status word
S (as command)	Stop move in progress abruptly
SADDR#	Set motor to new address
SILENT	Suppress PRINT messages main channel
SILENT1	Suppress PRINT messages second channel
SIZE=expression	Number of data entries in cam table
SLEEP	Initiate sleep mode main channel
SLEEP1	Initiate sleep mode second channel
STACK	Reset nesting stack tracking
SWITCH expression	Program execution control
T=expression	Assign torque value in torque mode
TALK	Enable PRINT messages on main channel
TALK1	Enable PRINT messages on main channel
TEMP	Temperature variable
TH	Sets high temperature set point
THD	Sets temperature fault delay
TWAIT	Pause program during a move
UA=expression	Set I/O A output
UAA	I/O A analog input value (0 to 1024)
UAI (as command)	Set I/O A to input
UAI (as input value)	I/O A input value variable
UAO (as command)	Set I/O A to output
UB=expression	Set I/O B output
UBA	I/O B analog input value (0 to 1024)
UBI (as command)	Set I/O B to input
UBI (as input value)	I/O B input value variable
UBO (as command)	Set I/O B to output
UC=expression	Set I/O C output
UCA	I/O C analog input value (0 to 1024)

UCI (as command)	Set I/O C to input
UCI (as input value)	I/O C input value variable
UCO (as command)	Set I/O C to output
UCP (as command)	Set I/O C to be a right limit input
UD=expression	Set I/O D output
UDA	I/O D analog input value (0 to 1024)
UDI (as command)	Set I/O D to input
UDI (as input value)	I/O D input value variable
UDM (as command)	Set I/O D to be a left limit input
UDO (as command)	Set I/O D to output
UE=expression	Set I/O E output
UEA	I/O E analog input value (0 to 1024)
UEI (as command)	Set I/O E to input
UEI (as input value)	I/O E input value variable
UEO (as command)	Set I/O E to output
UF=expression	Set I/O F output
UFA	I/O F analog input value (0 to 1024)
UFI (as command)	Set I/O F to input
UFI (as input value)	I/O F input value variable
UFO (as command)	Set I/O F to output
UG=expression	Set I/O G output
UGA	I/O G analog input value (0 to 1024)
UGA (as command)	Set I/O G to G synchronous function
UGI (as command)	Set I/O G to input
UGI (as input value)	I/O G input value variable
UGO (as command)	Set I/O G to output
UIA	Read Current (Amps = UIA/100)
UJA	Read Voltage (Volts = UJA/10)
UP	Upload user EEPROM program contents
UPLOAD	Upload user EEPROM readable program

V=expression	Set maximum permitted velocity
VLD	Sequentially load variables from data EEPROM
VST	Sequentially store variables to data EEPROM
WAIT=expression	Suspends program for number of PID samples
WAKE	Terminate sleep mode main channel
WAKE1	Terminate sleep mode second channel
WHILE expression	Conditional program flow command
X	Slow motor motion to stop
Z	Total system reset
Za	Reset current limit violation latch bit
Zb	Reset serial data parity violation latch bit
Zc	Reset communications buffer overflow latch bit
Zd	Reset math overflow violation latch bit
Zf	Reset serial comm framing error latch bit
Zl	Reset historical left limit latch bit
Zr	Reset historical right limit latch bit
Zs	Reset command scan error latch bit
Zu	Reset user array index access latch bit
Zw	Reset encoder wrap around event latch bit
ZS	Reset system latches to power-up state

APPENDIX D: SMARTMOTOR COMMANDS

The **SMI** software is a free download from Animatics' web site. The user still must have the proper cable(s) and power source for the SmartMotor being tested. Every SmartMotor has an ASCII interpreter built in so technically, it is possible to talk to the motor without the **SMI** software.

To download the software go to our web site (www.smartmotor.com, see right) and click on the ***Technical Support*** option on the top right side of the Animatics screen.

In the middle of the second screen ***Technical Support***, click on **SMI downloads.**

The third screen is legalese concerning how the software is used. **I Agree** must be selected before the software can be downloaded.

The next screen (below) should be **File Download**. Make sure **Save file to disk** is selected and click **OK**.

The **Save As** screen (below) should now be on screen. The small data window at the top of the screen, **Save In:**, should have a default folder already selected (possibly **TEMP** as it is here). Use this window to change drives. In the middle of the **Save As** screen is a large field with a list of files and folders. From here the **Save In:** folder can be changed by double clicking on a different folder or right clicking in an empty portion of the field and selecting **New** then **Folder** from the new menus and entering a different name. After the new folder has been created, double click on it to select it and continue on.

SmartMotor web site opening screen

SmartMotor web site Technical Support screen

Windows' File Download screen

Windows' Save As screen

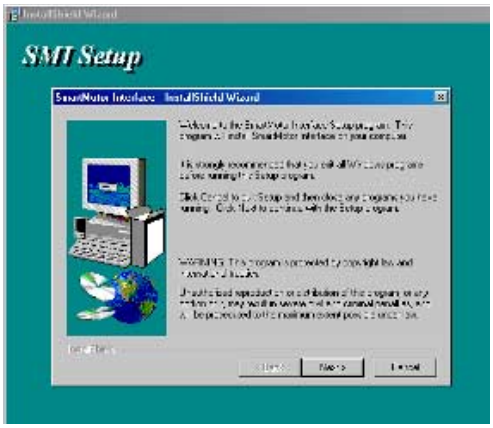
At the bottom of the **Save As** screen are two smaller data windows. The first window (**File name:**) is where the file can be renamed (don't change the **.exe** extension or the file won't work). Make sure the bottom window (**Save as type:**) has **Application** entered and then click on **Save**.

The last screen is the file download progress window. When it's finished the software can be installed. If the **Close this dialog box when download complete** check box is checked the window will close when the file has finished downloading.

The latest version of the SMI software is available at www.smartmotor.com.

If there are any Window's programs running, close them before installing the **SMI** software.

If the **SMI** software is being installed from the SmartMotor CD, insert the CD into the drive. The **SMI** installation program should automatically start. If not, run "**setup.exe**" from the root directory of the CD.



*The **SMI** installation setup window*

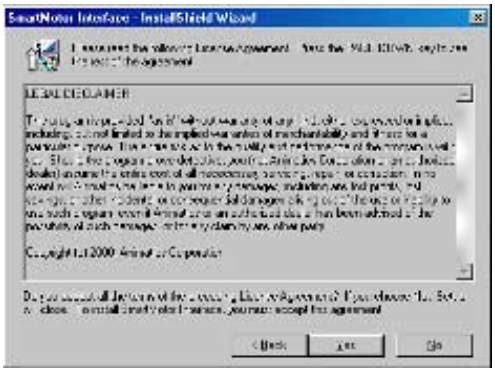
If the **SMI** software is being installed from the downloaded program run "**SMIsetup.exe**" from the directory the software was downloaded to.

Above, is the installation program's opening screen. Press the **Next** button to go to the next step.

License Agreement

This screen is the license agreement.

Read it carefully and if the terms are agreeable, click the **Yes** button to go on. If the **No** button is selected, the installation program will close and the software will not be installed. The terms must be agreed to before the softwate can be installed.



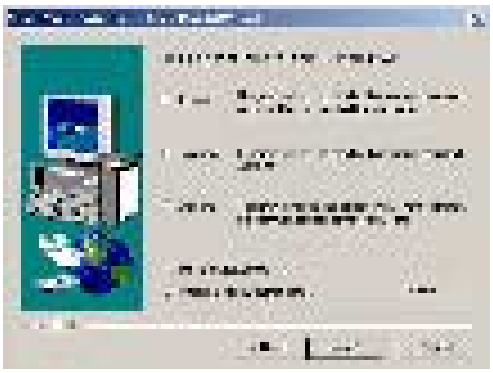
License agreement window

Type of installation

There are three types of installation:

Typical: This is the recommended option for most users. It installs all the common features of the **SMI** software.

Compact: This option installs the minimum required features. Since the typical installation uses very little hard disk space, this option isn't recommended.



Type of installation dialog window.

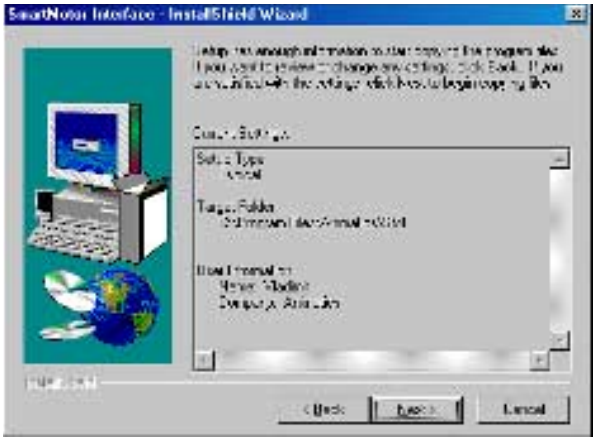
Custom: Select this option to choose only the features to be installed. If the **SMI** software needs to be installed in a directory other than the default directory shown in **Destination Directory** window, click the **Browse** button and select a different directory.

Click the **Next** button to go to the next step.

The confirmation window

The confirmation window

This window gives the user an opportunity to review the settings before copying the files. If anything needs to be changed click on the **Back** button to go to the desired window and change the settings. Click the **Next** button to begin copying files.



The Finish window

The Finish dialog window

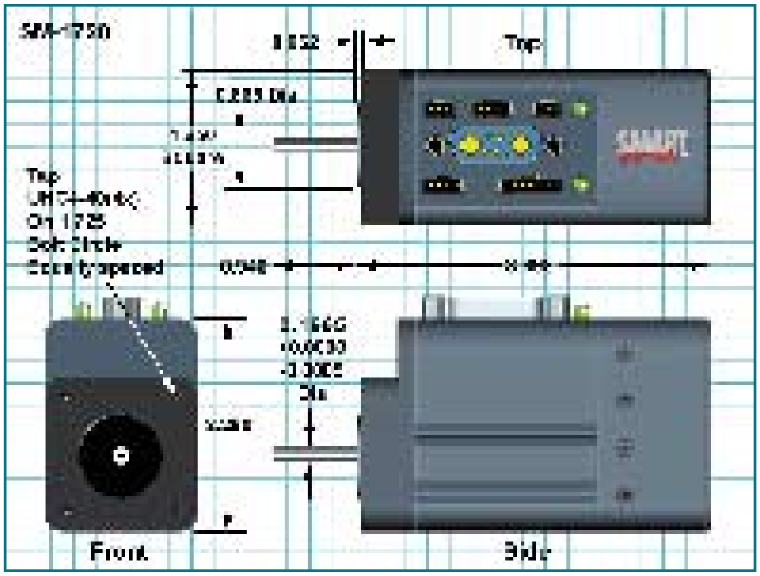
When the files are done copying, the final window (right) will be onscreen.

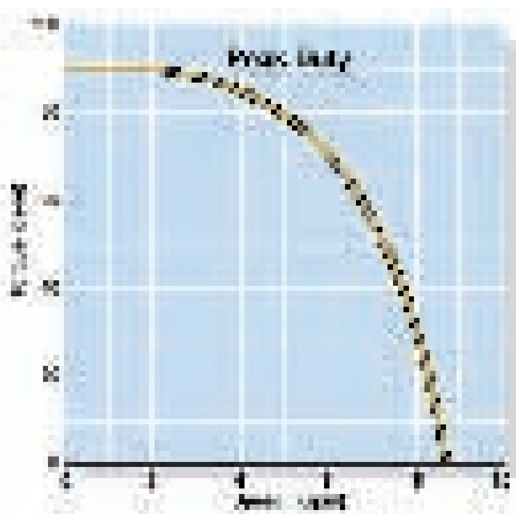
There are two choices in this window, **Yes, I want to restart my computer now** and **No, I will restart my computer later**. The computer must be restarted before the **SMI** software can be used. After the choice has been made, click the **Finish** button

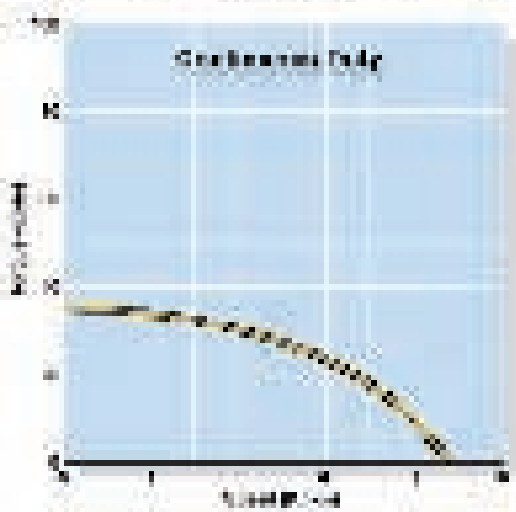
SmartMotor Connections

Before starting the **SMI** software make sure that the SmartMotor power and communication cables are properly connected.

SM1720M



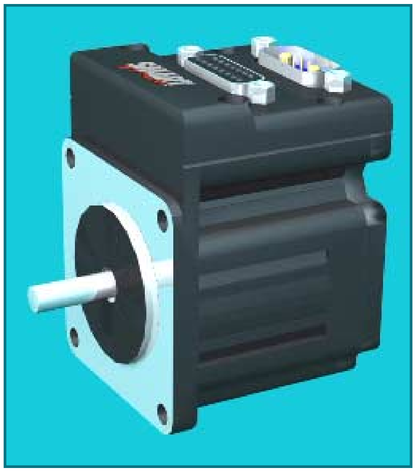


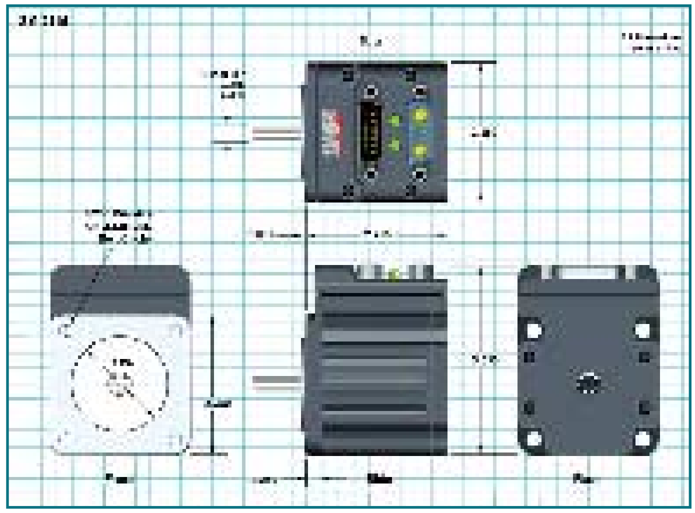


Continuous Torque	[illegible]	[illegible]
Peak Torque	[illegible]	[illegible]
Torque Constant	[illegible]	[illegible]
Nominal Continuous Power	[illegible]	[illegible]
[illegible] Speed	[illegible]	[illegible]
Voltage Constant	[illegible]	[illegible]
Winding Resistance	[illegible]	[illegible]
Encoder Resolution		[illegible]
Rotor Inertia	[illegible]	[illegible]
Poles		[illegible]
Slots		[illegible]
Shaft Diameter	in	[illegible]
Weight	lb	[illegible]
Length	in	[illegible]
Width	in	[illegible]
[illegible]		
[illegible]		◆
[illegible]		◆
[illegible]		◆
[illegible]		◆
[illegible]		◆
[illegible]		◆
[illegible]		
[illegible]		◆
[illegible]		
[illegible]		
CE Mark		◆
[illegible]		◆
[illegible]		
Expansion Bus		

Many free-wheeling vertical applications require a SHUNT to protect the SmartMotor from over voltage resulting from back-drive to the supply, especially if the supply is already very close to the SmartMotor's 48VDC input voltage limit.

Multiple axis applications enduring high current spikes should consider using Isolated RS-485 adapters. Order RS485-ISO for each motor and an RS232485 for the host end. Smaller motors such as the SM1720M with reasonably smooth or soft accelerations work very well in a simple RS-232 daisy chain.

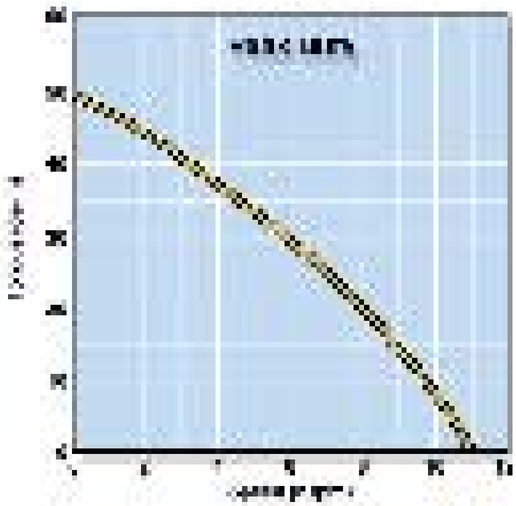


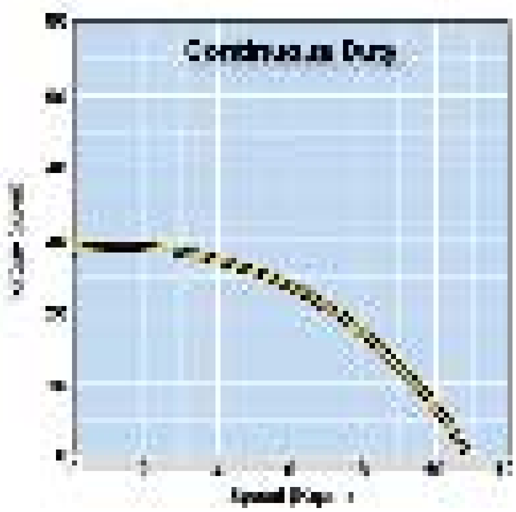


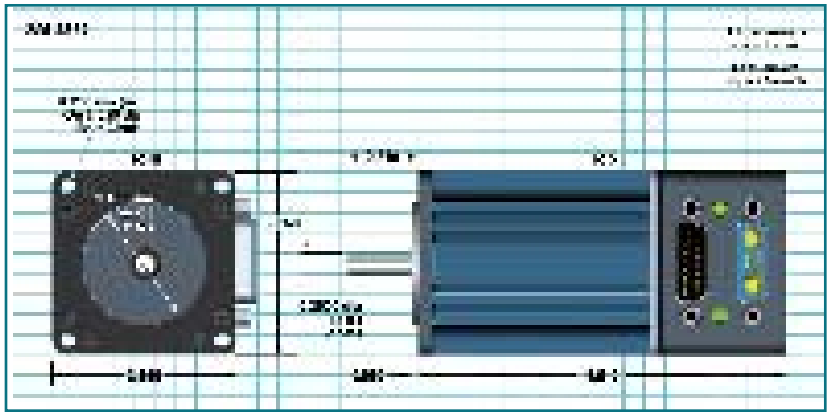
Continuous Torque	[illegible]	[illegible]
Peak Torque	[illegible]	[illegible]
Torque Constant	[illegible]	[illegible]
Nominal Continuous Power	[illegible]	[illegible]
Top Speed	[illegible]	[illegible]
Voltage Constant	[illegible]	[illegible]
Winding Resistance	ohms	[illegible]
Encoder Resolution		2000
[illegible]	[illegible]	[illegible]
Poles		[illegible]
Slots		[illegible]
Shaft Diameter	[illegible]	[illegible]
Weight	[illegible]	[illegible]
Length	inches	[illegible]
Width	inches	[illegible]
[illegible]		
Encoder Index Output		
Encoder Follow/Gearing		
Infinite Ratio Camming		
Removable Memory		
Non-Volatile Data Memory		
[illegible] RS-485		
[illegible] RS-485		
[illegible]		✦
[illegible]		
[illegible]		
CE Mark		✦
[illegible]		✦
[illegible]		✦
Expansion Bay		✦

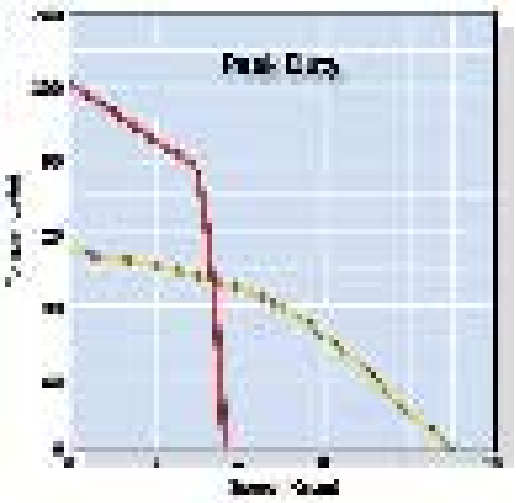
The SM2315 has two features that set apart from other SmartMotors. One, is the 0.250" hole in the back of the shaft. This hole is designed for a press fit with standard, under size 1/4 inch shafting and greatly facilitates customer rear shaft additions and modifications (PLEASE BE SURE NOT TO PRESS AGAINST THE BEARINGS).

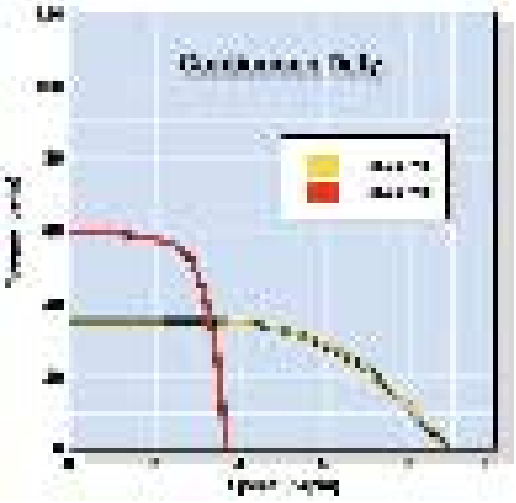
The second unique feature is an interior electronics expansion bay located just inside the back cover. This additional and internally connectorized space is designed to facilitate the addition of custom electronics. Consult the factory for more information on how to take advantage of this unique feature in your application.



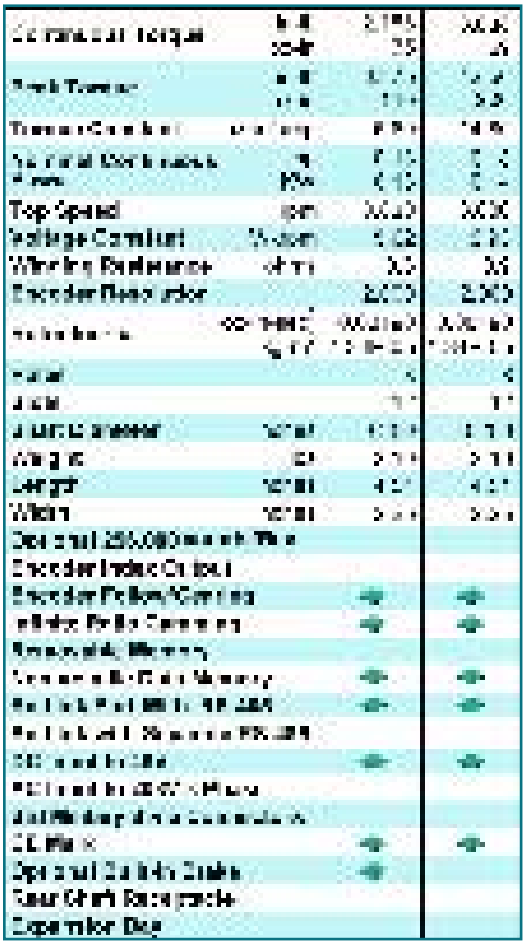




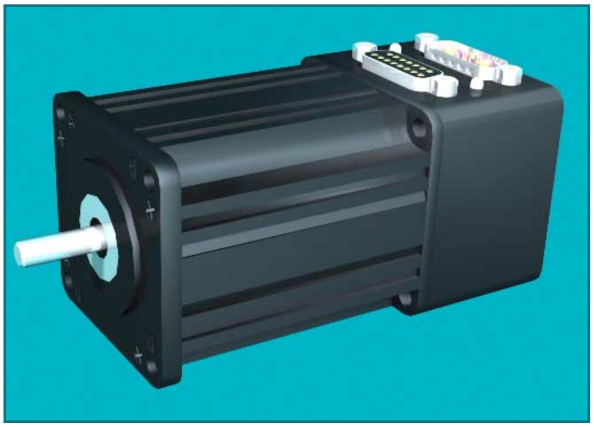


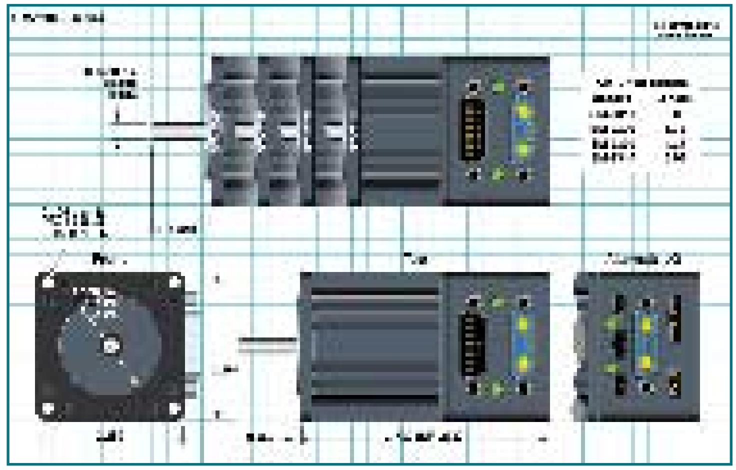


Many free-wheeling vertical applications require a SHUNT to protect the SmartMotor from over voltage resulting from back-drive to the supply, especially if the supply is already very close to the SmartMotor's 48VDC input voltage limit.



Multiple axis applications enduring high current spikes should consider using Isolated RS-485 adapters. Order RS485-ISO for each motor and an RS232485 for the host end. Smaller motors such as the SM23XX Series with reasonably smooth or soft accelerations work very well in a simple RS-232 daisy chain.

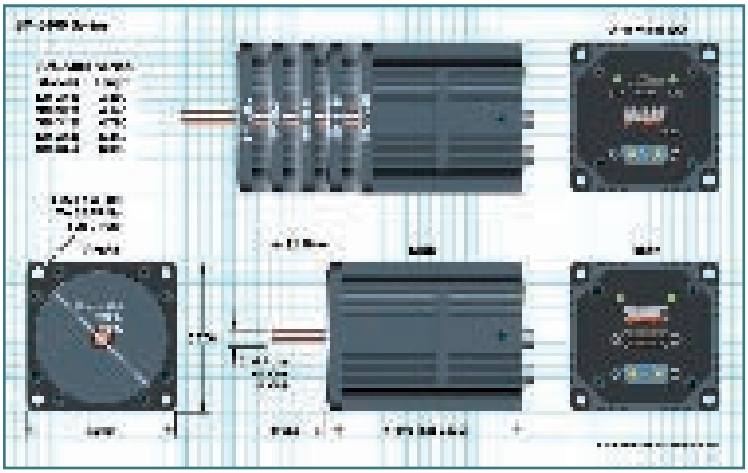




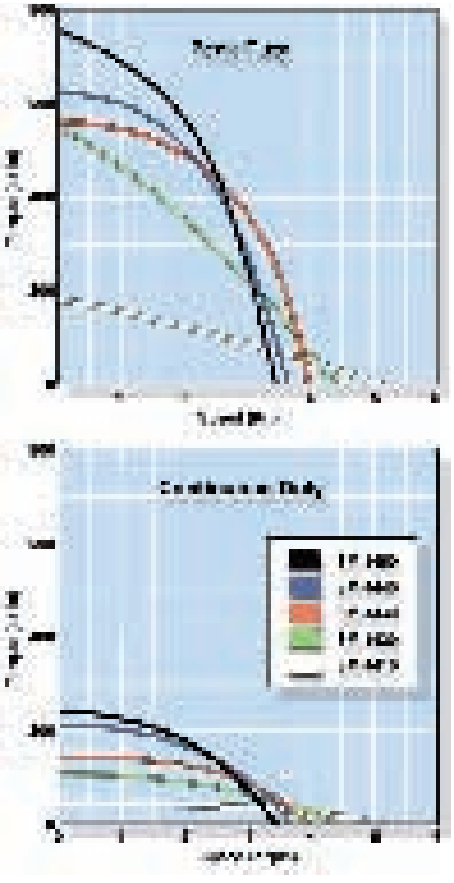
Continuous Torque	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Peak Torque	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Torque Constant	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Nominal Continuous	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Power	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
No Load Speed	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Voltage Constant	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Winding Resistance	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Encoder Resolution		[illegible]	[illegible]	[illegible]	[illegible]
Rotor Inertia	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Poles		[illegible]	[illegible]	[illegible]	[illegible]
Slots		[illegible]	15	15	15
Shaft Diameter	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Weight	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Length	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Width	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]		◆	◆	◆	◆
[illegible]		◆			
[illegible]		◆	◆	◆	◆
[illegible]		◆	◆	◆	◆
[illegible]					
[illegible]		◆	◆	◆	◆
[illegible]		◆	◆	◆	◆
[illegible]					
[illegible]		◆	◆	◆	◆
[illegible]					
Std Military Style Connectors					
CE Mark		◆	◆	◆	◆
Optional Built-in Brake		◆	◆	◆	◆
[illegible]					
[illegible]					

Many vertical applications require a SHUNT to protect the SmartMotor from over voltage resulting from back-drive to the supply, especially if the supply is already very close to the SmartMotor's 48VDC input voltage limit.

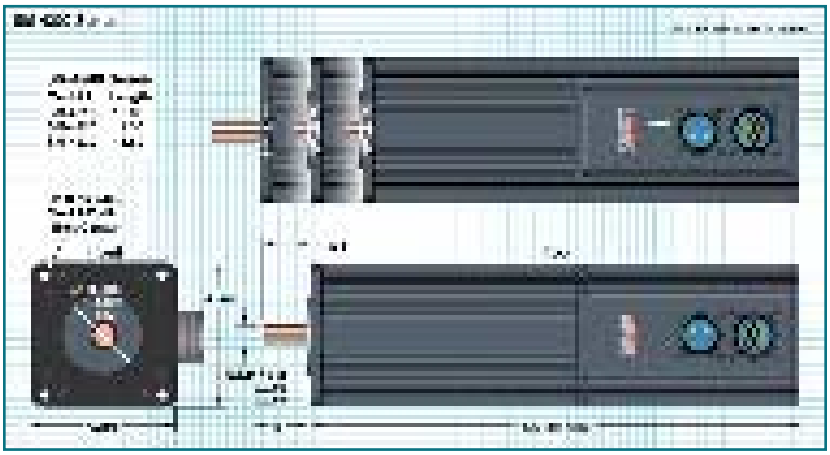
Multiple axis applications enduring high current spikes should consider using Isolated RS-485 adapters. Order RS485-ISO for each motor and an RS232485 for the host end.



Torque Constant						
Power Leads						

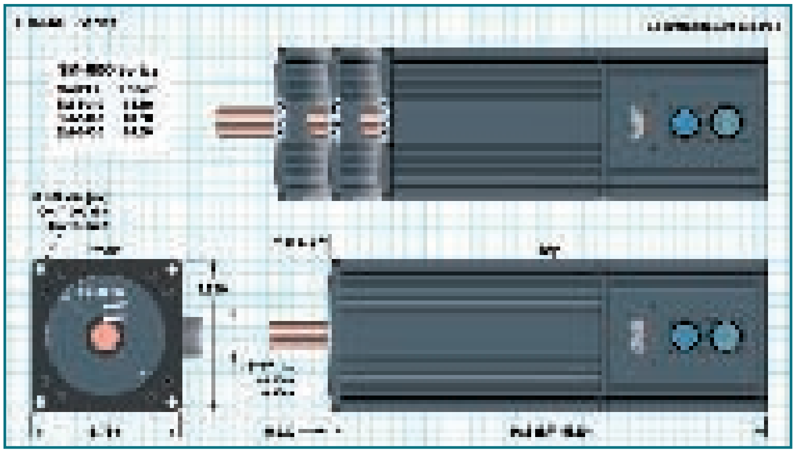


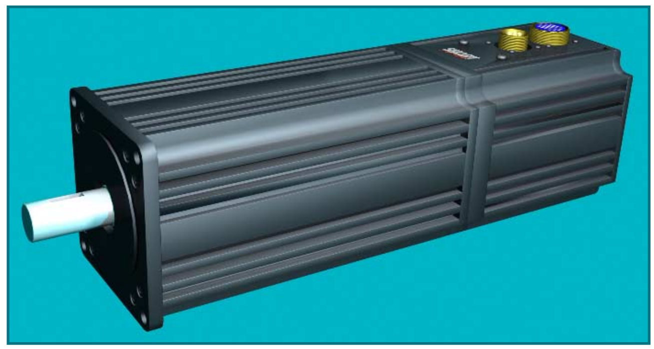
APPENDIX G: MOTOR SPECIFICATIONS
SM4210, SM4220, SM4230



SmartMotors are designed to operate within ambient temperatures ranging between zero and 70 degrees Centegrade. The ratings are for standard room temperature. Continuous torque output derates to zero as the ambient temperature spans from room temperature to 70C.

If the SmartMotor is working hard it will heat up, sometimes so much so as to be too hot to touch. That is normal. SmartMotor MTBF figures are calculated assuming the electronics are at the maximum temperature of 70C (worst case). These theoretical calculations put the SmartMotor MTBF beyond 100,000 hours. Actual field data, now normalized over thousands of installations, shows real SmartMotor average life to exceed the calculated estimates.





Continuous Torque	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]
Peak Torque	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]
Torque Constant	[illegible]	[illegible]	[illegible]	[illegible]
Nominal Continuous Power	[illegible]	[illegible]	[illegible]	[illegible]
	W	1.25	[illegible]	[illegible]
Top Speed	rpm	[illegible]	[illegible]	[illegible]
Voltage Constant	[illegible]	[illegible]	[illegible]	[illegible]
Winding Resistance	[illegible]	[illegible]	[illegible]	[illegible]
Encoder Resolution		4000	4000	4000
Rotor Inertia	[illegible]	[illegible]	[illegible]	[illegible]
	[illegible]	[illegible]	[illegible]	[illegible]
Poles		[illegible]	[illegible]	[illegible]
Slots		[illegible]	[illegible]	[illegible]
Shaft Diameter	inches	[illegible]	[illegible]	[illegible]
Weight	[illegible]	[illegible]	[illegible]	[illegible]
Length	inches	[illegible]	[illegible]	[illegible]
Width	inches	[illegible]	[illegible]	[illegible]
[illegible]				
Encoder Index Output		◆	◆	◆
[illegible]		◆	◆	◆
[illegible]		◆	◆	◆
[illegible]				
[illegible]				
[illegible]				
[illegible]		◆	◆	◆
[illegible]				
AC Input [illegible]		◆	◆	◆
[illegible]		◆	◆	◆
CE Mark				
Optional [illegible] Brake				
Rear Shaft [illegible]				
Expansion Bay				