Language reference table of contents

Language reference table of contents (continued)

Language reference introduction

The Smartmotor™ "Language Reference" lists each Smartmotor command in alphabetical order. Every command is described in exacting detail and shown in the context of a real-world example where it applies.

The commands are supplemented with a "Related Commands" section in the outside column that is designed to guide you to other pertinent commands and assure that you become aware of every resource the Smartmotor has to offer to address your specific need.

The examples are printed in a bold in a MORE STRUCTURED FONT to be quickly and unmistakably identified and interpreted. Comments are included and separated with a single quotation mark as they would be in your own programs.

You will almost certainly find the SmartMotor programmability the most powerful of any motion controller you have ever used. Any problem you may be facing will have many solutions to choose from. The key to successful application programming is knowing enough to choose the most elegant solution available.

Please let us know if you find any errors or omissions in this book so that we may improve it for future readers. Such notifications should be sent by e-mail with the words "Language Reference" in the subject line sent to: **info@animatics.com**. Thank you in advance for your contribution.

©2001, 2002 Animatics Corporation. All rights reserved

Animatics The SmartMotor Language Reference.

This book is furnished under license and may be used or copied only in accordance with the terms of such license. The content of this book is furnished for informational use only, is subject to change without notice and should not be construed as a commitment by Animatics Corporation. Animatics Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this book.

Except as permitted by such license, no part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Animatics Corporation.

Animatics, the Animatics logo, SmartMotor and the SmartMotor logo are all trademarks of Animatics Corporation. Windows, Windows 95/98, Windows 2000 Windows NT and Windows XP are all trademarks of Microsoft Corporation.

Contact Us:

Animatics Corporation
3200 Patrick Henry Dr.
Santa Clara, CA 95054
USA
Tel: 1 (408) 748-8721
Fax: 1 (408) 748-8725
www.animatics.com

! (exclamation point)

Pause Program Execution

Related Commands:

GETCHR

GETCHR1

APPLICATION: Program flow control

DESCRIPTION: Pauses Program Execution

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: Use **ENTER** key from host terminal

REPORT COMMAND: None

READ/WRITE: N/A

LANGUAGE ACCESS: Use only within a user program

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: All

DETAILED DESCRIPTION:

The command **!** suspends the user program until a properly terminated character or string is received through the SmartMotor™ serial port. As long as the SmartMotor is in command mode, the character or string received will be interpreted as a command.

The **!** command is useful when debugging new programs and stopping output streams from the motor at runtime. The **!** command doesn't affect the trajectory generator or a move in progress.

See sample code on next page:

Related Commands:

GETCHR

GETCHR1

EXAMPLE: (user debug output page with pause)

```
 a=10000000        'program parameter
O=a     'set axis origin
MP      'set buffered motion mode to Mode Position
A=100   'set buffered acceleration
V=4000  'set buffered maximum velocity
P=-a    'set buffered target position
b=50    'loop counter
c=0     'data set counter
GOSUB10 'call debug routine
G
WHILE b 'while b>0
    GOSUB10         'emit data set
    IF Bt==0    'exit if trajectory done
        BREAK
    ENDIF
    b=b-1       'decrement loop index
LOOP
GOSUB10 'emit final data set
END     'program terminate



C10
    c=c+1       'increment data set counter
    'NOTE PRINT(#13) sends a carriage return
    PRINT(#13,#13,"DATA SET ")
    Rc
    PRINT(#13,"Value of a       ",a)
    PRINT(#13,"Value of b       ",b)
    PRINT(#13,"Position         ")
    RP
    PRINT("Velocity         ")
    RV
    PRINT("Acceleration     ")
    RA
    PRINT("Position Error   ")
    RPE

      !                  'wait for ENTER from SMI terminal
RETURN
```

Related Commands:

Carriage Return

APPLICATION: Program flow control

DESCRIPTION: Single spaces placed between a series of user variables or commands

EXECUTION: Immediate

FIRMWARE VERSIONS: All

DETAILED DESCRIPTION:

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: Serial communications channel data

A single space character may be placed between a series of user commands in a single ASCII string as delimiter. If sent from a PLC or PC, the same space character can be used as an string terminating character.

When assigning values to sequential variables, use between assigned value and terminate sequence with an immediately following period.
The space character can also be used in PRINT command strings in like manner.

EXAMPLE as Delimiter for variable initialization:

```
n 7 2 8 56.         '(Note spaces and period)
```

equivalent:

```
n=7 o=2 p=8 q=56


t=6
aw[t] 63 44 98.   '(Note spaces and period)
```

equivalent:

```
aw[6]=63 aw[7]=44 aw[8]=98
```

EXAMPLE as Delimiter and Null Terminator in PRINT command:

```
PRINT("a=1 b=2 ")
'note space after b=2 as null terminator
```

equivalent:

```
PRINT("a=1 b=2",#13)
'note carriage return as null terminator
```

Note: When sending commands via serial port from a PC or PLC or other controller, a space character can be used as both a delimiter and a string terminator. It can be used equally and interchangeably with a carriage return as a string terminator.

Related Commands:

P

V

A

APPLICATION:	Direct Mode Position, Velocity, and Acceleration Data
DESCRIPTION:	Binary Packet Data
EXECUTION:	Immediate
CONDITIONAL TO:	Appropriate terminal
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	Serial communications channel data
UNITS:	Function byte + 32 bit binary packet
RANGE OF VALUES:	0x80000000 to 0x7FFFFFFF
TYPICAL VALUES:	0x80000000 to 0x7FFFFFFF
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	Version 3.2, firmware version G3 and higher

DETAILED DESCRIPTION:

Direct Mode commands always have the following five byte format: a single command byte, followed by four data bytes. There are three command bytes presently available in hex format:

0xFE	Commanded **P**osition Header Bit
0xFD	Commanded **V**elocity Header Bit
0xFC	Commanded **A**cceleration Header Bit

Note: Binary strings set Buffered Values!
To have them take effect, they must also be followed by a G command and a Null Terminator (Carriage Return or Space Character)

EXAMPLE:

Set Buffered target position to 100 **(P=100)**

0xFE 0x00 0x00 0x00 0x64

Set Buffered target position to -2 **(P=-2)**

0xFE 0xFF 0xFF 0xFF 0xFE

Related Commands:

P

V

A

Set Buffered target velocity to 10000 **(V=10000)**

0xFD 0x00 0x00 0x27 0x10

Set Buffered target velocity to -10000 **(V=-10000)**

0xFE 0xFF 0xFF 0xD8 0xF0

Set Buffered target acceleration to 1024 **(A=1024)**

0xFD 0x00 0x00 0x04 0x00

Note: A<0 is not valid.

Since a direct mode command is always in a fixed format, it doesn't require an end of line character. However, to have the buffered values take effect, the **G** character may be directly appended to the end of any direct mode command.

EXAMPLE:

Set Buffered target position to 100 and "Go" **(P=100 G)**

0xFE 0x00 0x00 0x00 0x64 0x47 0x20

Set Buffered target acceleration to 100 and "Go" **(A=100 G)**

0xFC 0x00 0x00 0x00 0x64 0x47 0x20

Keep in mind, Proper Mode commands must be set up prior to binary command strings in order to get predictable results. If Velocity Mode Is required, then first send MV followed by the associated binary commands.
This would then allow for fast changes in speed once in velocity mode.

Related Commands:

P

RP

@PE

@V

ENC0

ENC1

APPLICATION:	Monitor trajectory
DESCRIPTION:	Fetch Real-Time Encoder Position
EXECUTION:	Next **PID** sample
CONDITIONAL TO:	N/A
LIMITATIONS:	Expression value
REPORT COMMAND:	**RP**
READ/WRITE:	Read only
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	N/A
DEFAULT VALUE:	0 at power reset
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

@P is used to access the value of the primary encoder. This number may be called the current position or actual position. If the motor shaft moves the value of **@P** will be changed by the net number of encoder counts occurring during this shaft motion. The primary encoder is tracked at all times and is independent of the mode of operation of the SmartMotor™, or any error condition.

PRINT(@P) and **RP** would transmit an identical value if It were possible to execute both commands at the same time.

@P cannot be used to store a new value to a given shaft position; to change the point of origin for the encoder use the syntax **O=expression**. To set a desired target position use **P=expression**.

EXAMPLE:

```
A=100               'set buffered acceleration
V=40000             'set buffered velocity
MV                  'set to Mode Velocity
G                   'GO, start motion trajectory
WHILE @P<=5000      'wait until real time position
LOOP                'exceeds 5000 counts
PRINT("Position is above 5000",#13)
```

Note: @P follows the primary encoder used to close the loop. If you issue ENC1, it will follow an external encoder. Please see ENC0 and ENC1 for more details.

Related Commands:

E

@P

APPLICATION: Monitor trajectory

DESCRIPTION: Fetch Real-Time Position Error

EXECUTION: Next **PID** sample

CONDITIONAL TO: None

LIMITATIONS: Expression value

REPORT COMMAND: **RPE**

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Encoder counts

RANGE OF VALUES: Magnitude limited to user set value of **E**

TYPICAL VALUES: 0 to 32000

DEFAULT VALUE: 1000

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

Position Error is the difference in encoder counts between the desired trajectory position and the measured position. If the absolute value of **@PE** exceeds the user value **E,** the drive stage will turn off immediately setting both the **Bo** (Motor Off) and **Be** (Position Error) status bits will be set to 1, within that **PID** servo sample. When the servo is off, **@PE** reverts to zero since there is no longer a desired position.

PRINT(@PE) and **RPE** would transmit an identical value if it were possible to execute both commands at the exactly the same time.

Note: As acceleration, **A**, is increased, a larger value of **E** will be required.
E is unsigned but **@PE** may be positive or negative.

EXAMPLE:

```
E=1000      'set maximum position error permitted
A=100       'set buffered acceleration
V=3200000   'set buffered maximum velocity
P=12345678  'set buffered target position
G           'move to target
WHILE Bt    'while trajectory in progress
      IF @PE>800
            PRINT(#13,"WARNING)
            PRINT(#13,"Postion error close to limit")
      ENDIF
LOOP
```

Related Commands:

V

MV

RV

@P

@PE

PIDn

APPLICATION:	Monitor trajectory
DESCRIPTION:	Commanded PID Trajectory Velocity
EXECUTION:	Next **PID** sample
CONDITIONAL TO:	Calculated Trajectory
LIMITATIONS:	Expression value
REPORT COMMAND:	RV, PRINT(@V)
READ/WRITE:	Hardware read only
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Scaled encoder counts per **PID** sample (65536 scaled counts = 1 count)
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-3000000** to **3000000**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The function **@V** returns the present target trajectory velocity. Instead of returning the actual velocity, it tells you what the velocity is supposed to be. For the most part, this is the same as the actual velocity, for the simple reason that, if you are not at the right velocity, you are likely in position error. Similarly, if you observe the position error is not changing (see **@PE**), the present reported velocity is the exact velocity.

Equations for Real world Units:

Velocity (Encoder Counts/Sec) = @V x ***k***
Velocity (RPS) = @V x ***k*** / Encoder Resolution
Velocity (RPM) = @V x ***k*** / Encoder Resolution x 60

Where: Encoder Resolution = Encoder Counts per Revolution

and ***k***=0.0620876 for all standard SmartMotors™ <=v4.95

When in Position or Velocity Mode, **MP** or **MV**, the actual velocity is enforced by the **PID** feedback control to match the desired velocity computed by the trajectory generator.

If the position error (see **@PE**) is exactly constant, the actual velocity will exactly match the desired velocity over time, that is, macroscopically with respect to time.

(Continued on following page)

Related Commands:

V

MV

RV

@P

@PE

PIDn

While Accelerating, the position error may increase as a result of the physical velocity being less than the trajectory velocity. During the constant velocity slew phase, if position error were constant, physical velocity would equal the trajectory velocity on average.

Looking at time microscopically, within one **PID** sample, the limit of encoder measurement is one encoder count, a velocity granularity of **65536** scaled counts, per sample. This is in contrast to the macroscopic velocity, which has a granularity of one scaled count. In position or velocity mode, the macroscopic trajectory velocity with a granularity of 1 scaled count per sample is returned by **@V**.

In modes that do not generate a trajectory velocity, for example, torque mode, the velocity must be gleaned from changes in the encoder each Sample, so the microscopic value with a granularity of **65536** scaled counts per sample is returned by **@V**.

RV, **PRINT(@V)**, and the sequence **a=@V Ra** would transmit identical values, if it were possible to execute all three command sequences simultaneously.

To display the user-specified buffered maximum velocity value **V** (**V=expression**), as opposed to the present velocity, the sequences **a=V Ra** or equivalently **PRINT(V)** would be used.

EXAMPLE:

```
A=20                'set buffered acceleration
V=66500             'set buffered velocity
MV                  'Set to Velocity Mode
G                   'Begin Moving
WHILE @V<V          'wait for acceleration phase  to complete
LOOP
PRINT("Target Velocity has ben reached",#13)
```

Related Commands:

aa . . zzz

ab[index]

al[index]

aw[index]

APPLICATION:	General purpose data control
DESCRIPTION:	User signed 32 bit variables
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Versions prior to 4.00 only have variables **a . . . j**
REPORT COMMAND:	**Ra . . Rz**
READ/WRITE:	Read Write
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	Signed 32 bit Integer
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-2147483648** to **2147483647**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher
	Versions prior to 4.00 have 10 variables, **a . . j**

DETAILED DESCRIPTION:

The SmartMotor™ has three groups of pre-defined user variables. The first group consists of the variables **a** through **z**. They are general purpose Read/Write 32 bit signed integer variables that can be reported and used on either side of an equal sign in an equation.

The variables **a** thru **z** are stored in Dynamic RAM, meaning
Their values are lost when power is lost!

The value of any variable **a** through **z** variable is reported with the **R**, **PRINT( )** or **PRINT1( )** functions.

EXAMPLE:

```
Rg                   'Report the value of g to the primary serial port
PRINT("g=",g,#13)          'Print to the primary serial port.
PRINT1("g=",g,#13)         'Print to the secondary serial port.
```

SEE APPENDIX C
To describe the relationship between user assigned variables, ***aa*** *thru* ***zzz,*** *and variable arrays,* ***ab[ ]***, ***al[ ]*** *and* ***aw[ ]***

All 32 bit signed integer variables are limited to integer values between **-2147483648** to **2147483647**. Math operations that result in decimal values are truncated, or rounded down. If you assign or perform an operation that would normally result in a value outside of this range, the variable will "wrap," or take on the corresponding modulo. As an example, because of this, 2147483647+1=-2147483648. The result "wrapped around" to the negative extreme.

Related Commands:

aa . . zzz

al[index]

aw[index]

ab[index]

SEE APPENDIX C
*To describe the relationship between user assigned variables, **aa** thru **zzz,** and variable arrays, **ab[ ]**, **al[ ]** and **aw[ ]***

The following are other restrictions:

- If **a+b** exceeds 32 signed bits the operation **c=a+b** will produce a wrong result. No error flag is set.
- If **a-b** exceeds 32 signed bits the operation **c=a-b** will produce a wrong result. No error flag is set.
- If **a*b** exceeds 32 signed bits the operation **c=a*b** will produce a wrong result. The system flag **Bd** will bet set.

If one of these variables is used with a variable of another type, it will be appropriately converted. In technical jargon, the variable will be type cast. For example, in the equation where the variable on the left of the equal sign is a 16 bit one like **aw[4]**, all variables will be converted to 16 bit values and then operated on. Assigning the variable **aw[27] = y** directly stores the 16 least significant bits of **y** into **aw[27]**. The higher bits of the variable **y** are lost. Similarly, if the right hand variable is an 8 bit one like **ab[167]**, all variables will be converted to 8 bit values before being operated on. Conversely, if the left hand value is a 32 bit variable and the right hand side contains 16 bit variables, the 16 bit variables will be temporarily "upgraded" to 32 bits. In the equation **c=ab[4]-aw[7]**, both **ab[4]** and **aw[7]** are converted into 32 bit numbers before the subtraction occurs.

In the SmartMotor™ language, all user variables are written as lower case letters, while functions and commands have at least one upper case character. The term **a** is a general purpose variable, while **A** is the **A**cceleration function. Any user variable can be assigned a value with an equation, as discussed above, but can also be sequentially loaded by specifying the starting variable and the series of values to be loaded.

EXAMPLE:
Suppose the following code:

```
c=123        'assign the value of    123 to "c"
d=345        'assign the value of    345 to "d"
e=-599       'assign the value of   -599 to "e"
f=346        'assign the value of    346 to "f"
g=678678     'assign the value of 678678 to "g"
```

The Sequential loading method equivalent is as follows:

```
c 123 345 -599 346 678678.    'sequentially load data into
                              'variable c thru g
```

Note: The last number MUST BE followed by a "." period.

All user variables are initialized to the value of **0** at power up or upon execution of the system reset command **Z**. Other than by direct assignment, this is the only way that the SmartMotor sets all of the user variables to **0**. Issuing a **RUN** command does not perform this automatic initialization. For this reason, it is usually preferred to test a program, whether it is auto-execution or not, by power cycling the SmartMotor or issuing a system reset command **Z**.

Related Commands:

a . . z

ab[index]

al[index]

aw[index]

APPLICATION:	General purpose data control
DESCRIPTION:	User signed 32 variables
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	This data space is shared with **ab[ ]**, **aw[ ]**, **al[ ]** arrays, and coordinated motion (see mode **MD**)
REPORT COMMAND:	**Raa . . . Rzzz**
READ/WRITE:	Read Write
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	Signed 32 bit Integer
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-2147483648** to **2147483647**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

The SmartMotor™ has three groups of pre-defined user variables. The second and third group consists of the variables **aa** through **zz** and **aaa** through **zzz**. They are general purpose Read/Write 32 bit signed integer variables that can be reported and used on either side of an equal sign in an equation.

All variables **aa** thru **zzz** are stored in Dynamic RAM, meaning
Their values are lost when power is lost!

The value of any variable **aa** through **zzz** variable is reported with the **R**, **PRINT( )** or **PRINT1( )** functions.

SEE APPENDIX C
*To describe the relationship between user assigned variables, **aa** thru **zzz,** and variable arrays, **ab[ ]**, **al[ ]** and **aw[ ]***

EXAMPLE:

```
Rgg                   'Report the value of gg to the primary serial port
PRINT("gg=",gg,#13)      'Print to the primary serial port.
PRINT1("gg=",gg,#13)     'Print to the secondary serial port.
```

Unlike the variables set **a** through **z**, the variables **aa** through **zz** and **aaa** through **zzz** are overlaid with the variable arrays **ab[ ]**, **aw[ ]** and **al[ ]**.

As signed 32 bit variables, they are subject to the usual restrictions of signed digital words and values. The first bit is always a sign bit. They are limited to integer values between **-2147483648** to **2147483647**. Math operations that result in decimal values are truncated, or rounded down. If you assign or perform an operation that would normally result in a value outside of this range, the variable will "wrap", or take on the corresponding modulo. As an example, because of this, 2147483647+1=-2147483648. The result "wrapped around" to the negative extreme.

Related Commands:

a . . z

ab[index]

aw[index]

al[index]

Bit Overflow Status (Bd System Status bit):

- If **aa+bb** exceeds 32 signed bits the operation **cc=aa+bb** will produce a wrong result. No error flag is set.
- If **aa-bb** exceeds 32 signed bits the operation **cc=aa-bb** will produce a wrong result. No error flag is set.
- If **aa*bb** exceeds 32 signed bits the operation **cc=aa*bb** will produce a wrong result. The system flag, **Bd**, will be set.

If one of these variables is used with a variable of another type, it will be appropriately converted. In technical jargon, the variable will be type cast. For example, if a 16 bit variable like **aw[4]** is used, all variables will be converted to 16 bit values and then operated on. Assigning the variable **aw[27]=yy** directly stories the 16 least significant bits of **yy** to **aw[27]**. The higher bits of the variable **yy** are lost. Similarly, if the left hand variable is an 8 bit one like **ab[167]**, all variables will be converted to 8 bit values before being operated on. Conversely, if the left hand value is a 32 bit variable and the right hand side contains 16 bit variables, the 16 bit variables will be temporarily "upgraded" to 32 bits. In the equation **cc=ab[4]-aw[7]**, both **ab[4]** and **aw[7]** are converted into 32 bit numbers before the subtraction occurs.

EXAMPLE:

Suppose the following code:

```
cc=123      'assign the value of    123 to "cc"
dd=345      'assign the value of    345 to "dd"
ee=-599     'assign the value of   -599 to "ee"
ff=346      'assign the value of    346 to "ff"
gg=678678   'assign the value of 678678 to "gg"
```

The Sequential loading methode equivlent is as follows:

```
cc 123 345 -599 346 678678.   'sequentially load data into
                              'variable cc thru gg
```

Note: The last number MUST BE followed by a "." period.

All user variables are initialized to the value of **0** at power up or upon execution of the system reset command, **Z**. Other than by direct assignment, this is the only way the SmartMotor™ sets all of the user variables to **0**. Issuing a **RUN** command doesn't perform this automatic initialization. For this reason, it is usually preferred to test a program, whether it is auto-execution or not, by power cycling the SmartMotor or issuing a system reset command, **Z**.

Related Commands:

a . . z

aa . . zz

aaa . . zzz

aw[index]

al[index]

VST

VLD

SEE APPENDIX C
*To describe the relationship between user assigned variables, **aa** thru **zzz,** and variable arrays, **ab[ ]**, **al[ ]** and **aw[ ]***

APPLICATION:	General purpose data control
DESCRIPTION:	User signed 8 bit variables
EXECUTION:	Immediate
CONDITIONAL TO:	Index values 0 to 203
LIMITATIONS:	Index limited to number or sum or difference of any **a** . . **z** This data space is shared with variables **aa . . zz**, **aaa . . zzz**, arrays **aw[ ]** and **al[ ]**, and coordinated motion (**MD**).
REPORT COMMAND:	**Rab[index]**
READ/WRITE:	Read write
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	Signed 8 bit number
RANGE OF VALUES:	**-128 to 127**
TYPICAL VALUES:	**-128 to 127**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

The SmartMotor™ has 8, 16 and 32 bit array variable access. The 8 bit array takes the form of the variables **ab[index]**. These are general purpose 8 bit signed integer variables that can be reported, used on either side of an equation, and used with variables other than 8 bit. Like all user variables, they are always lower case, can be sequentially loaded, and are automatically initialized to zero at power up or reset. All arrays share memory space with the variables **aa** through **zz** and **aaa** through **zzz**.

The syntax of the 8 bit array is ab[index], which stands for array byte, and accepts an index value between **0** and **203**. This index can be specified explicitly or though another variable. For example, **ab[4]** refers to the fifth element in the 8 bit array, while **ab[n]** refers to the n'th element of the array, where the value of **"n"** must be between **0** and **203**.

The value of any array variable is reported with the **R**, **PRINT( )** or **PRINT1( )** functions.

EXAMPLE:

```
Rab[47]      'Report the value of ab[47] to the primary serial port
PRINT("ab[47]=",ab[47],#13)    'Print to the primary serial port.
PRINT1("ab[47]=",ab[47],#13)   'Print to the secondary serial port.
```

Related Commands:

a . . z

aa . . zz

aaa . . zzz

aw[index]

al[index]

VST

VLD

The **ab[ ]** array is classified as read write, meaning that they can be assigned a value, or can be assigned to some other variable or function. Another way of saying this, is these variables can be left or right hand values.

EXAMPLE:

```
ab[24]=ab[43]+ab[7]
```

The above is a valid equation, combining the contents of **ab[43]** and **ab[7]** and sending the total into **ab[24]**.

As signed 8 bit variables, they are subject to the usual restrictions of signed digital words and values. The first bit is always a sign bit. They are limited to integer values between **-128** and **127**. Math operations that result in decimal values are truncated, or rounded down. If you assign or perform an operation that would normally result in a value outside of this range, the variable will "wrap," or take on the corresponding modulo. As an example, because of this, 127+1=-128. The result "wrapped around" to the negative extreme.

Bit Overflow Status (Bd System Status bit):

- If **ab[1]+a** exceeds 32 signed bits the operation **c=ab[1]+a** will produce a wrong result. No error flag is set.
- If **a-ab[1]** exceeds 32 signed bits the operation **c=a-ab[1]** will produce a wrong result. No error flag is set.
- If **a*ab[1]** exceeds 32 signed bits the operation **c=a*ab[1]** will produce a wrong result. The system flag, **Bd**, will be set.

If one of these variables is used with a variable of another type, it will be appropriately converted (the variable will be type cast).

EXAMPLE:

In the equation where the variable on the left of the equal sign is an 8 bit one like **ab[4]**, all variables will be converted to 8 bit values and then operated on. Assigning the variable **ab[27]=al[m]** directly stores the 8 least significant bits of **al[m]** into **aw[27]**. The higher bits of the variable **al[m]** are lost. Conversely, if the left hand value is a 32 bit variable and the right hand side contains both 8 bit and 16 bit variables, the 8 bit and 16 bit variables will be temporarily "upgraded" to 32 bits. In the equation **al[3]=ab[4]-aw[7]**, both **ab[4]** and **aw[7]** are converted into 32 bit numbers before the subtraction occurs.

In the SmartMotor™ language, all user variables are written as lower case variables, while functions and commands have at least one upper case character. The term **ab[i]** is a general purpose variable, while **A** is the acceleration function. Any user variable can be assigned a value with an equation, but can also be sequentially loaded by specifying the starting variable and the series of values to be loaded.

(Continued on following page)

Related Commands:

a . . z

aa . . zz

aaa . . zzz

aw[index]

al[index]

VST

VLD

SEE APPENDIX C
*To describe the relationship between user assigned variables, **aa** thru **zzz,** and variable arrays, **ab[ ]**, **al[ ]** and **aw[ ]***

(Continued from preceding page)

EXAMPLE:

```
ab[6] 123 34 67 34 127.
```

Loads sets **ab[6]** equal to **123**, **aw[7]** to **34** and so forth, ending with **127** loaded into **ab[10]**. The command syntax requires a space between the leading variable and each subsequent value. The function is terminated by a period.

All user variables are initialized to the value of **0** at power up or upon execution of the system reset command **Z**. Other than by direct assignment, this is the only way that the SmartMotor™ sets all of the user variables to **0**. Issuing a **RUN** command does not perform this automatic initialization. For this reason, it is usually preferred to test a program, whether it is auto-execution or not, by power cycling the SmartMotor or issuing a system reset command **Z**.

The **aa** through **zz** and **aaa** through **zzz** variables share the same physical memory as part of the **ab[ ]**, **aw[ ]** and **al[ ]** arrays. That is, if you set **aaa=123456**, you will find **al[0]** has the same value, regardless of what you set it to before. Similarly, the values of **ab[0]** through **ab[3]** and **aw[0]** and **aw[1]** will have values that correspond to the individual 8 bit bytes and 16 bit words that are part of **aa.**

Related Commands:

a . . z

aa . . zz

aaa . . zzz

ab[index]

aw[index]

VST

SEE APPENDIX C
*To describe the relationship between user assigned variables, **aa** thru **zzz,** and variable arrays, **ab[ ]**, **al[ ]** and **aw[ ]***

APPLICATION: General purpose data

DESCRIPTION: User signed 32 bit variables

EXECUTION: Immediate

CONDITIONAL TO: The value of index must be between **0** and **50**

LIMITATIONS: This data space is shared with variables **aa** . . **zz**, **aaa . . zzz**, **arrays ab[ ]** and **aw[ ]**, and coordinated motion (see **MD**)

REPORT COMMAND: **Ral[index]**

READ/WRITE: Read write

LANGUAGE ACCESS: Assignment, expressions and conditional testing

UNITS: Signed 32 bit number

RANGE OF VALUES: **-2147483648** to **2147483647**

TYPICAL VALUES: **-2147483648** to **2147483647**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

The SmartMotor™ has 8, 16 and 32 bit arrays. The 32 bit array takes the form of the variables al[index]. These are general purpose 32 bit signed variables that can be reported, used on either side of an equation, and used with variables other than 32 bit. Like all user variables, they are always lower case, can be sequentially loaded and are automatically initialized at power up or reset. All arrays share memory space with the variables **aa** through **zz** and **aaa** through **zzz**.

The syntax of the 32 bit array is **al[index]** (**al** stands for **array long**) and accepts an index value between **0** and **49**. This index can be specified explicitly or though another variable.

EXAMPLE:

al[4] refers to the fifth element (count begins with zero) in the 32 bit array.

The value of any array element **al[ ]** is reported with the **R**, **PRINT( )** or **PRINT1( )** functions. For example to send the value of variable **al[47]** out the primary serial port, u se the command **Ral[47]** or **PRINT(al[47],#13)**. To send the value of the variable **al[37]** out serial port 1, use **PRINT1(al[37],#13)**.

The **al[ ]** array is classified as read write, meaning that they can be assigned a

Related Commands*:*

a . . z

aa . . zz

aaa . . zzz

ab[index]

aw[index]

VST

value, or can be assigned to some other variable or function. Another way of saying this, though more cryptically technocratic, is that these variables can be left or right hand values.

EXAMPLE:

```
al[24]=al[43]+al[7]
```

is a valid equation, combining **al[43]** and **al[7]** and sending the total into **al[24]**.

As signed 32 bit variables, they are subject to the usual restrictions of signed digital words and values. The first bit is always a sign bit. They are limited to integer values between **-2147483648** to **2147483647**. Math operations that result in decimal values are truncated, or rounded down. If you assign or perform an operation that would normally result in a value outside of this range, the variable will "wrap," or take on the corresponding modulo. As an example, because of this, 2147483647+1=-2147483648. The result "wrapped around" to the negative extreme.

Bit Overflow Status (Bd System Status bit):

- If **al[1]+a** exceeds 32 signed bits the operation **c=al[1]+a** will produce a wrong result. No error flag is set.
- If **a-al[1]** exceeds 32 signed bits the operation **c=a-al[1]** will produce a wrong result. No error flag is set.
- If **a*al[1]** exceeds 32 signed bits the operation **c=a*al[1]** will produce a wrong result. <u>The system flag, **Bd**, will be set.</u>

If one of these variables is used with a variable of another type, it will be appropriately converted (the variable will be type cast).

EXAMPLE:

In the equation where the variable on the left of the equal sign is a 16 bit one like **aw[4]**, all variables will be converted to 16 bit values and then operated on. Assigning the variable **aw[27]=al[m]** directly stores the 16 least significant bits of **al[m]** into **aw[27]**. The higher bits of the variable **al[m]** are lost. Similarly, if the left variable is an 8 bit one like **ab[167]**, all variables will be converted to 8 bit values before being operated on. Conversely, if the left value is a 32 bit variable and the right side contains both 8 and 16 bit variables, both 8 and 16 bit variables will be temporarily "upgraded" to 32 bits. In the equation **al[3]=ab[4]-aw[7]**, both **ab[4]** and **aw[7]** are converted into 32 bit numbers before the subtraction occurs.

In the SmartMotor™ language, all user variables are written as lower case variables, while functions and commands have at least one upper case character. The term **al[i]** is a general purpose variable, while **A** is the **A**cceleration function. Any user variable can be assigned a value with an equation, as discussed above, but can also be sequentially loaded by specifying the starting variable and the series of values to be loaded. ***(Continued on following page)***

Related Commands:

a . . z

aa . . zz

aaa . . zzz

ab[index]

aw[index]

VST

VLD

SEE APPENDIX C
To describe the relationship between user assigned variables, ***aa*** *thru* ***zzz,*** *and variable arrays,* ***ab[ ]***, ***al[ ]*** *and* ***aw[ ]***

(Continued from preceding page)

EXAMPLE:

al[6] 123 345 567 346 678678.

The above loads sets **al[6]** equal to **123, al[7]** to **345** and so forth, ending with **678678** loaded into **al[10]**. The command syntax requires a space between the leading variable and each subsequent value. The function is terminated by a period.

All user variables are initialized to the value of **0** at power up or upon execution of the system reset command **Z**. Other than by direct assignment, this is the only way that the SmartMotor™ sets all of the user variables to **0**. Issuing a **RUN** command does not perform this automatic initialization. For this reason, it is usually preferred to test a program, whether it is auto-execution or not, by power cycling the SmartMotor or issuing a system reset command, **Z**.

The **aa** through **zz** and **aaa** through **zzz** variables share the same physical memory as part of the **ab[ ]**, **aw[ ]** and **al[ ]** arrays. That is, if you set **aaa=123456**, you will find that **al[0]** has the same value, regardless of what you set it to before. Similarly, the values of **ab[0]** through **ab[3]** and **aw[0]** and **aw[1]** will have values that correspond to the individual 8 bit bytes and 16 bit words that are part of **aa.**

Related Commands:

a . . z

aa . . zz

aaa . . zzz

al[index]

ab[index]

VST

VLD

APPLICATION: General purpose data

DESCRIPTION: User signed 16 bit data variables

EXECUTION: Immediate

CONDITIONAL TO: Index values 0 to 101

LIMITATIONS: This data space is shared with variables **aa . . zz, aaa . . zzz**, arrays **ab[ ]** and **al[ ]**, and coordinated motion.

(see **MD**).

REPORT COMMAND: **Raw[index]**

READ/WRITE: Read write

LANGUAGE ACCESS: Assignment, expressions and conditional testing

UNITS: Signed 16 bit number

RANGE OF VALUES: **-32768** to **32767**

TYPICAL VALUES: **-32768** to **32767**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

The SmartMotor™ has 8, 16 and 32 bit arrays. The 16 bit array takes the form of the variables **aw[index]**. These are general purpose 16 bit signed variables that can be reported, used on either side of an equation, and used with variables other than 16 bit. Like all user variables, they are always lower case, can be sequentially loaded and are automatically initialized at power up or reset. All arrays share memory space with the variables **aa** through **zz** and **aaa** through **zzz**.

The syntax of the 16 bit array is **aw[index]**, which stands for array word, and accepts an index value between **0** and **101**. This index can be specified explicitly or though another variable.

APPENDIX C
(Page ?)

*uses tables to describe the relationship between user assigned variables, **aa** thru **zzz,** and variable arrays, **ab[ ]**, **al[ ]** and **aw[ ]***

EXAMPLE:

aw[4] refers to the fifth element in the 16 bit array

aw[i] refers to the (**I+I**)th element of the array, where the value of **i** must be between **0** and **101**.

The value of any array element **aw[ ]** is reported with the **R**, **PRINT( )** or **PRINT1( )** functions. For example to send the value of variable **aw[47]** out the primary serial port, use the command **Raw[47]** or **PRINT(aw[47],#13)**. To send the value of the variable **aw[37]**out serial port 1, use **PRINT1(aw[37],#13)**.The **aw[ ]** array is classified as read write, meaning that they can be assigned a value, or can be assigned to

Related Commands:

a . . z

aa . . zz

aaa . . zzz

al[index]

ab[index]

VST

VLD

some other variable or function. Another way of saying this, though more cryptically technocratic, is that these variables can be left or right hand values.

EXAMPLE:

```
aw[24]=aw[43]+aw[7]
```

The above is a perfectly valid equation, taking **aw[43]** and **aw[7]** and stuffing the sum into **aw[24]**.

As signed 16 bit variables, they are subject to the usual restrictions of signed digital words and values. The first bit is always a sign bit. They are limited to integer values between **-32768** and **32767**. Math operations that result in decimal values are truncated, or rounded down. If you assign or perform an operation that would normally result in a value outside of this range, the variable will "wrap," or take on the corresponding modulo. As an example, because of this, 32767+1=-32768. The result "wrapped around" to the negative extreme.

Bit Overflow Status (Bd System Status bit):

- If **aw[1]+a** exceeds 32 signed bits the operation **c=aw[1]+a** will produce a wrong result. No error flag is set.
- If **a-aw[1]** exceeds 32 signed bits the operation **c=a-aw[1]** will produce a wrong result. No error flag is set.
- If **a*aw[1]** exceeds 32 signed bits the operation **c=a*aw[1]** will produce a wrong result. <u>The system flag, **Bd**, will be set.</u>

If one of these variables is used with a variable of another type, it will be appropriately converted. In technical jargon, the variable will be type cast. For example, in the equation where the variable on the left of the equal sign is an 8 bit one like **ab[4]**, all variables will be converted to 8 bit values and then operated on. Assigning the variable **aw[27]=al[m]** directly stores the 16 least significant bits of **al[m]** into **aw[27]**. The higher bits of the variable **al[m]** are lost. Conversely, if the left value is a 32 bit variable and the right side contains 16 bit variables, the 16 bit variables will temporarily revert to 32 bits. In the equation **al[3]=ab[4]-aw[7]**, both **ab[4]** and **aw[7]** are converted into 32 bit numbers before the subtraction occurs.

In the SmartMotor™ language, all user variables are written as lower case variables, while functions and commands have at least one upper case character. The term **aw[i]** is a general purpose variable, while **A** is the **A**cceleration function. Any user variable can be assigned a value with an equation, as discussed above, but can also be sequentially loaded by specifying the starting variable and the series of values to be loaded.

(Continued on following page)

Related Commands:

a . . z

aa . . zz

aaa . . zzz

al[index]

ab[index]

VST

VLD

APPENDIX C
(Page ?)

*uses tables to describe the relationship between user assigned variables, **aa** thru **zzz,** and variable arrays, **ab[ ]**, **al[ ]** and **aw[ ]***

(Continued from preceding page)

EXAMPLE:

```
aw[6] 123 345 567 346 31868.
```

The above loads sets **aw[6]** equal to **123**, **aw[7]** to **345** and so forth, ending with **31868** loaded into **aw[10]**. The command syntax requires a space between the leading variable and each subsequent value. The function is terminated by a period.

All user variables are initialized to the value of **0** at power up or upon execution of the system reset command **Z.** Other than by direct assignment, this is the only way that the SmartMotor™ sets all of the user variables to **0**. Issuing a **RUN** command does not perform this automatic initialization. For this reason, it is usually preferred to test a program, whether it is auto-execution or not, by power cycling the SmartMotor or issuing a system reset command **Z**.

The **aa** through **zz** and **aaa** through **zzz** variables share the same physical memory as part of the **ab[ ]**, **aw[ ]** and **al[ ]** arrays. That is, if you set **aaa=123456**, you will find that **al[0]** has the same value, regardless of what you set it to before. Similarly, the values of **ab[0]** through **ab[3]** and **aw[0]** and **aw[1]** will have values that correspond to the individual 8 bit bytes and 16 bit words that are part of **aa**.

Related Commands:

D
E
G
MP
MV
PID*n*
P
S
V
X
F=1

APPLICATION:	Trajectory control
DESCRIPTION:	Set buffered acceleration
EXECUTION:	Buffered pending **G** command
CONDITIONAL TO:	**E, G**, **V**, **PID*n***
LIMITATIONS:	Must not be negative Effective value is rounded down to next even number
REPORT COMMAND:	**RA**
READ/WRITE:	Read write
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	Scaled encoder counted/sample^2
RANGE OF VALUES:	**0** to **2147483647**
TYPICAL VALUES:	**0** to **5000**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

Setting the buffered A value determines the acceleration that will be used by subsequent position or velocity moves to calculate the required trajectory. Changing A during a move will not alter the current trajectory unless a new G command is issued.

To set acceleration, take the desired acceleration in **Rev/S²**, multiply it by **7.91** for 500 line encoder motors and 15.82 for 1000 line encoder motors. Then set **A** equal to it (the integer portion).

Acceleration is pre-scaled by **65,536 (256*256)** and may range from **0** to **2,147,483,648**; the default value is zero. **A** is buffered by **G**. It should also be understood that since **A** is bit shifted once to the right by the extended **PID** filter loop, odd values for **A** will be reduced by **1** in operation. An **A=1** command will have the same effect as **A=0**, you won't go anywhere.

Equations for Real world Units:

$$\text{Acceleration (Encoder Counts/Sec}^2) = A \times \boldsymbol{k}$$
$$\text{Acceleration (RPS}^2) = A \times \boldsymbol{k} / \text{Encoder Resolution}$$
$$\text{Acceleration (RPM}^2) = A \times \boldsymbol{k} / \text{Encoder Resolution} \times 60$$

Where: Encoder Resolution = Encoder Counts per Revolution

and ***k***=252.63236 for all standard SmartMotors™ <=v4.95

Related Commands:

D

E

G

MP

MV

PID*n*

P

V

X

F=1

EXAMPLE:

```
MP        'Set Mode Position
A=5000    'Set Acceleration
P=20000   'Set Absolute Position
V=100000 'Set Velocity
G     'Start Motion
```

EXAMPLE:

```
A=100         'set buffered acceleration
V=750         'set buffered velocity
MV            'set buffered velocity mode
G             'Start motion
```

ADDR

Set Motor Address

Related Commands:

SADDR

APPLICATION:	Serial communications control
DESCRIPTION:	Motor address
EXECUTION:	N/A
CONDITIONAL TO:	Firmware >= 4.15, Use "SADDR=" for <4.15
LIMITATIONS:	N/A
REPORT COMMAND:	**PRINT(ADDR), (variable)=ADDR R(variable)**
READ/WRITE:	Read/Write above version 4.15
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	Address
RANGE OF VALUES:	**0** to **100**
TYPICAL VALUES:	**1** to **100**
DEFAULT VALUE:	**0** on power-up until assigned
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

SmartMotors™ are designed to be used as much in multiple axis systems as in single axis ones. For that reason, they have been afforded the ability to be uniquely addressed. This is done with the **ADDR=expression** command (not available in versions below 4.15. Use the **SADDR#** command). For example **ADDR=5** or **SADDR5** both set the motor's address to be **5**. **ADDR** is a read write function, so it can also be used to access the address of the current SmartMotor.

Using **ADDR** within a program permits an identical program stored in different motors to differentiate between motors and provide individual runtime controls.

```
SWITCH ADDR
      CASE 1              ' motors 1,2 and 3 "GO"
      CASE 2
      CASE 3 G
            BREAK
      CASE 4 S            ' motor 4  "STOP"
ENDS                      ' Start motion (or stop)
```

Note: ADDR=# syntax DOES NOT work with v4.40 SM2315 series motors! SADDR# syntax must be used to assign the address.

AIN{address}{input}

Analog Input from I/O Device

Related Commands:

AOUT

DIN

DOUT

UAA

APPLICATION:	Input command (use with Anilink device)
DESCRIPTION:	Fetch 8 bit analog input byte
EXECUTION:	Immediate AniLink byte read
CONDITIONAL TO:	N/A
LIMITATIONS:	Port = **A** .. **H** and Input = 1, 2, 3, or 4
REPORT COMMAND:	**RAIN{address}{input}**
READ/WRITE:	Read only
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Numerical value
RANGE OF VALUES:	**0** to **255**
TYPICAL VALUES:	**0** to **255**
DEFAULT VALUE:	**255** in absence of peripheral device
FIRMWARE VERSIONS:	ALL

All seven SmartMotor™ I/O points also serve as direct Analog inputs.

DETAILED DESCRIPTION:

The SmartMotor™ communicates with optional expansion cards such as the AIO-100 and AIO110 AniLink cards through the **AIN{address}{input}** command, where **address** refers to the address of the Anilnk card and **input** refers to the input channel of the Anilink card. The address is given as a character between **A** and **H**, while the input is between **1** and **4**. See the AIO-100 User Manual for specific details.

The **AIN{address}{input}** returns an unsigned 8 bit value, ranging from **0** to **255**, linearly corresponding to the analog voltage on the specified input channel. A return of **0** corresponds to 0 volts and **255** to 5 volts. If the specified card is not present or the connected is not present, the function will return a value of **255**.

The **AIN{address}{input}** function is read only. It cannot be used on the left side of an equation, but only on the right.

The value of the **AIN{address}{input}** function can be reported through the primary serial port with the **PRINT(AIN{address}{input},#13)** and **AIN{address}{input}** functions. To transmit the value through serial channel 1 use **PRINT1(AIN{address}{input},#13)**.

EXAMPLE:

```
x=AINA1    'Assign analog value of Port A input 1 to "a"
```

Please refer to the associated Users Manuals for specifics about each optional Analog I/O card.

Related Commands:

RAMPS

T

MT

APPLICATION:	Amplifier control
DESCRIPTION:	Sets maximum allowed PWM to motor windings
EXECUTION:	Next **PID** sample
CONDITIONAL TO:	N/A
LIMITATIONS:	Must not be negative
REPORT COMMAND:	**RAMPS**
READ/WRITE:	Read write
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	**1/1023** of maximum PWM permitted
RANGE OF VALUES:	0 to **1023**
TYPICAL VALUES:	**1000**
DEFAULT VALUE:	**1000**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

Referencing against a hard stop this way can eliminate an additional switch and cable.

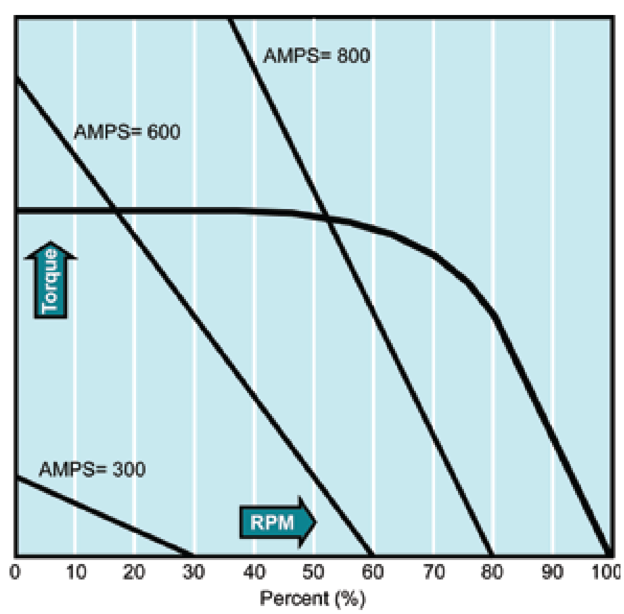
The **AMPS** command effectively limits both the continuous torque and speed of the SmartMotor™.

To set the SmartMotor to use maximum available PWM, issue the command **AMPS=1023**. Setting **AMPS=0** limits PWM to 0 thereby preventing any output torque. To conceptually understand what happens when you use values between **0** and **1023**, consider the following torque-speed diagram:

The **AMPS** function essentially cuts the torque-speed characteristic of the motor by slicing off the part of the curve to the right of the **AMPS** line. Note that there are some values of **AMPS** that will limit top speed but not peak torque. The slope of the line is highly dependent on the voltage of the power source.

AMPS is often used to limited torque and speed.

AMPS torque-speed diagram



AMPS has no effect in torque mode (**MT**, **T**) . In this mode, the value of **T** controls the commanded torque of the motor, without limitation by **AMPS**.

AOUT{address},{value}

Analog Output to I/O Device

Related Commands:

AIN

DIN

DOUT

APPLICATION: Output command (use with Anilink device)

DESCRIPTION: Output analog byte to Anilink peripheral port

EXECUTION: Immediate AniLink byte write

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: None

READ/WRITE: Write only

LANGUAGE ACCESS: Unlatched output value, to recall, create shadow variable

UNITS: Numerical value

RANGE OF VALUES: **0** to **255**

TYPICAL VALUES: **0** to **255**

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

When using an optional AniLink Analog I/O Card such as AIO-100 or AIO-110, an 8-bit (0-255) output voltage can be specified. Adjustments on the card allow the user to set the upper and lower limits, and therefore the range, anywhere between zero and Full scale output voltage. The examples assume the voltage reference inputs are set to full scale, zero and 5 VDC such as for the AIO-100.

EXAMPLES:

```
AOUTC128    'Output 2.5V Mod: C
```

Use a comma when using a variable:

```
a=128       'Set any variable
AOUTC,a     'Output to port
```

See the appendix for information about the use of the Ani-Link AIO-100 analog I/O expansion module and associated AniLink chip set.

The syntax of the command is **AOUT{address},{value}** sends a byte value to the associated AniLink peripheral card. The "address" of the AIO-100 card is a character between **A** and **H**, and is set on the card by three jumpers. The value is a number between **0** and **255**. If the value is **0**, the output voltage is the minimum value. If it is **255**, the voltage is maximum.

Please refer to the associated Users Manuals for specifics about each optional Analog I/O card.

Related Commands:

RW

RPW

Z

Za

ZS

APPLICATION: Monitor Motor status

DESCRIPTION: Over current detected state

EXECUTION: Historical, latched by **PID** sample

CONDITIONAL TO: Hardware Detection

LIMITATIONS: None

REPORT COMMAND: **Rba, RW(bit 14), RPW(bit14)**

READ/WRITE: Read only. To reset , issue **Za or ZS**

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Binary bit

RANGE OF VALUES: 0 or 1

TYPICAL VALUES: 0

DEFAULT VALUE: 0

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

The SmartMotor™ firmware checks each **PID** Sample to see whether or not a Peak-Over current condition exists. The set point is in hardware and depends on the model motor and drive stage. If the set point is reached, the system flag **Ba** is to **1**. Once an over-current has been detected, the SmartMotor will shut the amplifier off for several servo samples in attempt to reduce the peak load and then turn back on to try to complete its commanded motion. If the position error exceeds the allowable following error **E**, during the off state, the servo will get a Following Error (Be status Bit) .

The Ba bit is not reset until either a power reset, a **Z**, **ZS**, or **Za** command is issued.

Note: in non-PLS firmware motors, a "G" will reset the Ba bit.

If **Ba** flag is regularly found to be set there may be a problem. This typically indicated that the motor is undersized in the peak range. Please verify the motor is correctly "sized" for the presently assigned task.

IF the Ba bit is set every machine cycle, try lowering acceleration,. If it is still set very cycle, there could be a large moment of inertia mismatch.

The AMPS command has no effect on the Ba bit. It only effects continuous current, not peak current.

EXAMPLE: (sub component of system check routine)

```
IF Ba                           'If Peak over Current is detected
      PRINT("OVER CURRENT")     'inform host
      Za                        'clear over current state latch
ENDIF
```

Related Commands:

CHN

CHN0

CHN1

OCHN

Z

Zb

ZS

APPLICATION: Monitor Serial Communications

DESCRIPTION: Serial communications parity error detected state

EXECUTION: Historical, latched by serial communications

CONDITIONAL TO: Channel 0 or channel 1 open with Even or Odd parity

LIMITATIONS: None

REPORT COMMAND: **RBb**, **RCHN** (bit3), **RCHN0** (bit3), **RCHN1** (bit3)

READ/WRITE: Read only. To reset to zero issue **Zb** command

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0**

DEFAULT VALUE: **0** Not applicable to default No parity

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

The firmware checks for and flags any communications parity error event by setting **Bb** to **1**. If such an error occurs, an error recovery routine can be implemented at the discretion of the user. In practice, unless the environment is electrically noisy, this error is unlikely. Any data or syntax error due to noise is potentially dangerous in a motion control environment; please take appropriate precautions.

Parity only has relevance when the serial protocol includes parity checking. To include parity checking, the open channel command **OCHN** parity parameter must specify either even parity (E) or odd parity (O). The default is no parity (N), in which case there is no parity bit transmitted over the serial channel to check. If ignore parity (I) is specified as the parity parameter, there is a parity bit included with every data character, but it is not checked.

EXAMPLE: (sub component of system check routine)

```
OCHN(RS4,1,E,9600,1,8,C)      'open RS485 channel 1
IF Bb
      PRINT("SERIAL PARITY ERROR")
      Zb                'clear Parity Error status bit
ENDIF
```

Related Commands:

CHN

CHN0

CHN1

Z

Zc

ZS

OCHN

RCHN0

RCHN1

APPLICATION: Monitor Serial Communications

DESCRIPTION: Serial Communications Receive Buffer overflow occurred

EXECUTION: Historical, latched by Buffer Overflow detection

CONDITIONAL TO: Serial port buffer overflow

LIMITATIONS: None

REPORT COMMAND: **RBc**, **RCHN** (bit0), **RCHN0** (bit0), **RCHN1** (bit0)

READ/WRITE: Read only. To reset to zero issue **Zc** command

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 3.00 and higher

DETAILED DESCRIPTION:

Version 4.00 and higher motors have a software serial receive buffer maintained by the firmware. The buffer length is 16 characters. If a motor receives serial characters faster than the command interpreter can read them, the buffer will eventually overflow, and **Bc** is set to **1**. An error routine can be written to recover from such a failure.

In any serial daisy chain link, if characters are transmitted to the motors with no intermission between characters, the motors can get behind, eventually overflowing the motors' input buffer. The generally accepted solution is to put a delay between characters, between commands, or between long blocks of characters. In the case of the SmartMotor™, the above does not normally happen because most applications have naturally-occuring intervals between commands or groups of commands.

EXAMPLE: (sub component of system check routine)

```
IF Bc
   PRINT("SERIAL OVERFLOW")    'inform host
   Zc               'clear overflow state latch
ENDIF
```

Related Commands:

Zd

ZS

RW

RPW

APPLICATION:	Monitor expression evaluation math overflow
DESCRIPTION:	Math product overflow, value out of range
EXECUTION:	Historical, immediate
CONDITIONAL TO:	Software detects value out of range
LIMITATIONS:	None
REPORT COMMAND:	**RBd**, **RW** (bit 11), **RPW** (bit 11)
READ/WRITE:	Read only. To reset to zero issue **Zd** or **Zs** command
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	**0** or **1**
TYPICAL VALUES:	**0**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	3.00 and higher

DETAILED DESCRIPTION:

Bd is set to 1 by a math multiplication out of range condition (>32Bit signed integer) or an out of range Mode Follow Ratio result (>256).

The SmartMotor™ employs 32 bit signed integer calculations for all math functions. If, for example, **a*b** results in a magnitude greater than 31 binary bits, the **Bd** system flag is set to **1**. You can possibly avoid this by scaling the numbers, performing calculations in a different order, or using different method of calculation.

EXAMPLE:

Try this following product on your own hand held calculator and observe the result. Then try the same calculation using a motor.

```
Zd                 'reset error flag
zz=123456789
aa=987654321
f=aa*zz
Rf        <Response to host will be -67153019>
RBd       <Response to host will be 1>
```

Notice that even the sign of the product is incorrect.

Related Commands:

Zd

ZS

RW

RPW

EXAMPLE:

Mode Follow with Ratio permits the shaft to respond with a user defined scaling gain to the external encoder input. There is a limit to the magnitude of the gain such that

-256 < GAIN < 256

The system flag **Bd** is set if this **GAIN** restriction is violated.

The flag is set immediately after executing the **MFR** command.

```
Zd               'reset error flag
MFMUL=256 'Multiplier for incoming encoder counts
MFDIV=1          'Divisor for incoming encoder counts
RBd              'Response to host 0
MFR              'Calculate Mode Follow Ratio
RBd              'Response to host 1
```

The MFMUL parameter cannot exceed 256 * MFDIV.

Note: The Bd bit will only go out of range on multiplication of two numbers, not addition. In other words, IF you add two numbers and the result exceeds +/-2^{31} in magnitude, the number will be bit rolled over.

Example:

```
a=2140000000
ZS
b=a+a
Rb                 -14967296
```

Under the above condition even though the value of "b" is not correct, the Math overflow bit was not set.

Related Commands:

Ze

ZS

G

E

RW

RPW

APPLICATION: Monitor trajectory for error

DESCRIPTION: Position error declared

EXECUTION: Historical, immediate

CONDITIONAL TO: Position error exceeded **E** value during trajectory move

LIMITATIONS: Torque modes have no position error

REPORT COMMAND: **RBe**, **RS** (bit 5), **RW** (bit 5), **RPW** (bit 5)

READ/WRITE: Read only. Reset to issuing a **G** command

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The Be status bit indicates the detection of a position error. Each and every **PID** sample, the magnitude of the measured position error is compared to the allowable following error (**E)** value set by the user. If this value is exceeded, the servo will be immediately turned off, The **Bo** bit will be set to **1**, The **Bt** bit will be set to **0**, and **Be** will be set to **1 all at the same time**. If issued, **RMODE** will return an "E".

This condition is reset by:

* Issuing a G in non-PLS

* Issuing Ze or ZS (PLS firmware only).

EXAMPLE: (sub component move monitor routine)

```
TWAIT              'wait for trajectory in progress
                          'to complete
IF Be              'unsuccessful, position error?
      PRINT("POSITION ERROR")       'inform host
ENDIF
```

Note: an extended period of peak over current may result in a position error due to the fact that an over current condition will cause a reduction in power to the motor thereby causing it to fall behind possibly enough to exceed E (maximum allowable position error)

If a motor continuously gets a Position Error no matter what, check for loss of drive power, increased load or locked load.

Communications Framing Error Status Bit

Related Commands:

CHN

CHN0

CHN1

Z

Zf

ZS

APPLICATION: Monitor serial communications

DESCRIPTION: Serial communication framing error detected

EXECUTION: Historical, latched by serial communication receive

CONDITIONAL TO: Hardware detection

LIMITATIONS: None

REPORT COMMAND: **RBf**, **RCHN** (bit 1), **RCHN0** (bit 1), **RCHN1** (bit 1)

READ/WRITE: Read only. Reset to zero using command

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Bf indicates whether a framing error has been detected. Every serial byte received by the SmartMotor™ is checked to see if it has the correct start and stop bits, or "frame." If not, **Bf** is set to **1**. If such an error occurs, the error recovery routine is at the discretion of the user. In practice, unless the environment is electrically noisy, this error is unlikely. Any data error or syntax error due to noise is potentially dangerous in a motion control environment; please take appropriate precautions.

Note: A framing error can occur with slightly mismatched baud rates between two devices as well. SmartMotors meet the IEEE specification of baud rate +/-10%. If baud rates exceed that range between two devices, a framing error is likely to occur.

EXAMPLE: (sub component of system check routine)

```
IF Bf
      PRINT("SERIAL FRAMING ERROR") 'inform host
      Zf           'Reset framing error flag
ENDIF
```

Related Commands:

TEMP

TH

THD

Z

OFF

RW

RPW

APPLICATION: Monitor motor error state

DESCRIPTION: Hardware motor overheat state

EXECUTION: Real time, set after thermal delay (**THD**)/reset each **PID** sample

CONDITIONAL TO: Motor temperature, temperature set point (**TH**), temperature set point dead band, thermal delay (**THD**)

LIMITATIONS: None

REPORT COMMAND: **Real time:** **RBh**

Historical: **RS (bit 6), RW (bit 6), RPW (bit 6)**

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

There are two mechanisms in the SmartMotor™ that can indicate excessive heat. The first is a temperature sensor, while the second is an **RMS** current monitor. The former is a direct measurement of heat, while the latter predicts that an overheat will occur. In either case, **Bh** will be set to **1**.

With continuous heavy loads all motors will generate heat. If the heat sinking or ventilation is inadequate, eventually the motor will overheat. If this situation repeatedly occurs it may mean that the motor does not have enough power for the assigned task (motor sizing inadequate) or excessive resistance (friction) to motion is occurring. In this event, please check your overall motion system.

The overheat temperature limit is adjustable by the user by the **TH** command, but cannot exceed 70º Celsius (optional 85º). The amount of time that the temperature is allowed to stay at or above this temperature is set by the **THD** function. If the temperature stays at or above the **TH** value for longer than **THD** servo samples, the amplifier will turn off, **Bh** will be set to **1**, the motor off bit **Bo** set to **1** and the trajectory bit is cleared to **0 ALL at the same time!**. If issued, **RMODE** will return "O" meaning the drive stage is off. The SmartMotor will reject any command to start motion until the temperature has fallen 5º Celsius below the trip point.

Note: If power is removed and restored and temperature is <5 degrees below the set point, the motor will be allowed to move. This however can lead to damage if it is done repeatedly.

Related Commands:

TEMP

TH

THD

Z

OFF

RW

RPW

The **RMS** current monitor continuously calculates the equivalent Root-Mean-Square current of the amplifier. If the **RMS** current is too high for longer than **THD** servo samples, the amplifier will turn off, **Bh** will be set to **1**, the motor off bit **Bo** set to **1** and the trajectory bit cleared to **0 ALL at the same time!**. If issued, **RMODE** will return "O." The SmartMotor™ will reject any motion commands for approximately 10 milliseconds. The biggest difference between the two overheat mechanisms described will be that, if the **RMS** current monitor detects and overheat, the SmartMotor may not physically feel hot.

Once **Bh** is set to **1**, the historical overheat flag is latched when read by **RW**, **RS** or accessing **S**. If the overheat condition no longer exists, **Bh** will be reset to zero upon reporting (**RS**, **RW**) or accessing the **S** value.

EXAMPLE: (sub component of system check routine)

```
IF Bh
      IF TEMP>69
            PRINT("MOTOR TOO HOT")          'inform host
            GOSUB123                  'deal with condition
      ELSE
            PRINT("RMS Over Current Trip")
            GOSUB123                  'deal with condition
      ENDIF
ENDIF
```

EXAMPLE:

Test to measure approximate shut down time - not very accurate but illustrates **TH**, **THD**, and **TEMP.**

```
PRINT(#13,"Default value of TH = ",TH)   'default=70
PRINT(#13,"Motor Temperature = ",TEMP)
PRINT(#13,"START MOTION")
A=222
V=44444
MV
G
THD=32000    'THD default = 12000 or 3 seconds
                   ' units are PID samples
TH=TEMP-5    'Force an over heat condition
                   ' units are degrees Centigrade
                   ' TH maximum setting is 70
a=CLK
WHILE Bh==0 LOOP
WHILE Bt LOOP
b=CLK
PRINT(#13,"Servo OFF after ",b-a," PID samples")
```

Related Commands:

Bx

I

RI

F=

APPLICATION: Monitor Index Latching

DESCRIPTION: Hardware index position available state.

EXECUTION: Set upon hardware index latched

CONDITIONAL TO: Hardware index level detected high and prior index value read, F command and Port G.

LIMITATIONS: Latched until index value read

REPORT COMMAND: **RBi**, **RS** (bit3), **RW** (bit3), **RPW** (bit3)

READ/WRITE: Read only. Reset to zero by reading or assigning index value

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: Any legal encoder value

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

When enabled, the **Bi** flag is set to 1 when the internal encoder Z pulse (index mark) is detected. The value of the primary encoder in the servo sample that the index is captured is stored in the "**I**" system register WITHIN 5 microseconds of the time it was captured!

While **Bi** is **1**, the microprocessor is prohibited from making another index capture. If the captured value is read or accessed via accessing the **I** register via **RI of <variable>=I**, the **Bi** flag will be reset to zero and the ability to capture the index is again enabled.

The commands **RI** and **PRINT(I,#13)** report the captured index reading through the primary serial channel. **PRINT1(I,#13)** reports through the channel **1** serial port. Any of these command sequences reset the **Bi** flag to zero. Assignments such as **variable=I** likewise assign the captured value and reset the **Bi** flag to zero. If **Bi** is zero at the time the **I** value is accessed, the previously captured index value is again returned.

EXAMPLE: (simple homing)

```
MV                 'set buffered velocity mode
A=10               'set buffered acceleration
V=-4000            'set low buffered maximum velocity
G                  'start slow motion profile
WHILE Bm==0        'travel until negative limit reached
  i=I              'clear and arm index capture
LOOP
X                  'decelerate to a stop
P=I                'go back to index
G                  'start motion
TWAIT              'wait till end of trajectory
O=0                'set origin at index
```

Related Commands:

Bx

I

RI

F=

EXAMPLE: (Fast Index Find)

```
 MP                 'set buffered velocity mode
 A=1000             'set fast acceleration
 V=4000000          'set fast velocity
 D=2100             'set relative distance just beyond
                    'one shaft turn
 i=I                'clear and arm index capture
 O=0                'force change to position register
 G                  'start fast move
 TWAIT              'wait till end of trajectory
 P=I                'go back to index
 G                  'start motion
 TWAIT              'wait until end of trajectory
 O=0                'set origin at index
```

Index used as High Speed Position Capture:

When enabled via F=1024 (v4.95 or later firmware) the **Bi** flag is set to 1 when Port G I/O pin gets driven to zero. This happens within 5 microseconds of Port G going low. As a result Port G can be used to capture position for high speed registration applications

EXAMPLE: (Fast Position Capture)

```
 UGI                'Set Port G as Input Port
'Set F command flags
al[0]=64      'set value to enable C2 interrupt call
              '(C2 gets called when Port G grounded)
al[1]=1024    'set value to enable Index Position capture
              'to be triggered from Port G
  F=al[0]+al[1]
V=100000      'Set Velocity
A=100         'Set Acceleration
MV            'Set to Velocity Mode
G             'Start Moving
END

C2 'This routine gets called automatically when Port G goes low
      PRINT("Port G grounded when",#13)
      PRINT("position=",@P,#13)
RETURNI
```

SAMPLE TERMINAL SCREEN OUTPUT FROM ABOVE CODE:

(Port G repeatedly grounded)

```
Port G grounded when
position=226076
Port G grounded when
position=257022
Port G grounded when
position=271849
Port G grounded when
position=279430
Port G grounded when
position=295069
```

Related Commands:

RCKS

RW

RPW

LOAD

UPLOAD

VST

APPLICATION:	EEPROM data validation
DESCRIPTION:	EEPROM Checksum Failure State
EXECUTION:	Historical, set on eeprom data check
CONDITIONAL TO:	**RCKS**, **VST( )**, or **RES** calibration data check
LIMITATIONS:	Stored EEPROM program is **SMX** formatted
REPORT COMMAND:	**RBk**, **RW** (bit 15), **RPW** (bit 15)
READ/WRITE:	Read only, reset by **RCKS** and first post reset **RES** command
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	**0** or **1**
VALUE BY STATE:	**0**= valid EEPROM user program checksum and valid **VST( )** **1** = Invalid EEPROM user program checksum, or invalid **VST( )**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

Bk indicates whether a user program checksum write error has been detected. If **Bk** is **1**, either the user program and/or program header has been corrupted. You should not run the program in the SmartMotor™. This can occur if communications connection was lost or corrupted during a download of a program. **Bk** is reset to zero by a power reset, **Z**, and a valid (pass) checksum is detected via **RCKS.**

RCKS scans the entire program including header and returns two 6-bit checksums followed by a "P" (pass) or "F" (fail) at the end. If **RCKS** reports a failure, **Bk** is set to **1**. **RCKS** sends its value through the primary serial port.

EXAMPLE: (commands issued and responses from terminal screen)

```
RCKS          000049 0025E0 P
RBk           0
```

The **VST( )** command also has the capability to set Bk to 1. **VST( )** performs a read operation after every byte it writes to the User Data EEPROM; if the read byte is not the same as what was sent, the flag **Bk** is set to **1**.

Related Commands:

Bm
Bp
Br
LIMH
LIML
LIMD
LIMN
RS
RPW
RW
UCI
UCP
UCO
UDI
UDM
UDO
SLE
SLD
SLP
SLN
ZI
ZS

APPLICATION:	Monitor left limit switch
DESCRIPTION:	Left limit latch
EXECUTION:	Historical, sampled each **PID** update until latched
CONDITIONAL TO:	**LIMH. LIML, UDI, UDO, UDM**
LIMITATIONS:	None
REPORT COMMAND:	**RBI**, **RS** (bit 2), **RW** (bit 2), **RPW** (bit 2)
READ/WRITE:	Read only. Reset by **ZI**, **ZS**, **RS**, **RW**, **RPW**, assignment or printing of **S**
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	**0** or **1**
VALUE BY STATE:	**0**= Left/negative limit has not been active
	1= Left/negative limit has been active
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher firmware motors

DETAILED DESCRIPTION:

BI is the historical left limit flag. If the left limit is found to be active during any servo sample, **BI** is set to **1**, and remains **1** until reset by the user. In addition, the motion will stop and the motor will either servo or turn the amplifier **OFF**, depending on the value of the **F** function. The historical left/negative limit flag **BI** provides a latched value in case the limit may have already been reached and overpassed but is not at presently active.

The real time left/negative limit flag is **Bm**, which only remains set to **1** while the signal level on the user pin D is active. Whenever **Bm** is set to **1, BI** is set to one. The polarity of the signal that is considered active is determined by commands **LIMH** (Active High-To-Stop) and **LIML** (active Low-To_Stop) in all non-PLS firmware motors. PLS firmware motors are always Active High asserted.

If the pin's function is assigned to being general purpose I/O by use of the **UDI** or **UDO** commands, neither **Bm** nor **BI** will be affected by the pin state. Changing pin states will not elicit limit behavior from the motor. It will be necessary to issue the **UDM** command to assign the pin's function to being a limit switch, for the pin to again elicit limit behavior, including the setting of **BI**.

(Continued on next page)

Related Commands:

Bm

Bp

Br

LIMH

LIML

LIMD

LIMN

RS

RPW

RW

UCI

UCP

UCO

UDI

UDM

UDO

SLE

SLD

SLP

SLN

ZI

ZS

In non-PLS firmware motors, Bl is reset to zero under the following conditions:

1. When the **S** status byte is accessed for assignment
2. or reported via **RS**, **PRINT(S,#13)** or **PRINT1(S,#13)**
3. or directly reset with **Zl** and **ZS**.
4. or a **G** command is issued AND the **Bm** bit is not set.

In PLS firmware motors, Bl is reset to zero under the following conditions:

By issuing either **Zl** and **ZS**.

Example code:

```
IF Bm
      PRINT("LEFT LIMIT PRESENTLY ACTIVE")
ELSEIF Bl
      PRINT("LEFT LIMIT PREVIOUSLY CONTACTED")
ELSE
      PRINT("LEFT LIMIT NEVER REACHED")
ENDIF
```

Hardware Travel Limit Overview				Status Bits		Command to Clear Historical Bit	Command to Disable Travel Limit Input	Command to Enable Travel Limit Input
Port	Pos/Neg	Plus/Minus	Left/Right	Real Time	Historical			
Port C	Positive	PLUS	RIGHT	**Br**	**Bp**	**Zr, or ZS**	**UCI or UCO**	**UCP**
Port D	Negative	MINUS	LEFT	**Bl**	**Bm**	**Zl, or Zs**	**UDI or UDO**	**UDM**

Bm

Real-Time Left-Limit Status Bit

Related Commands:

Bl
Bp
Br
LIMH
LIML
LIMD
LIMN
RS
RPW
RW
UCI
UCP
UCO
UDI
UDM
UDO
SLE
SLD
SLP
SLN
Zl
ZS

APPLICATION:	Monitor left/negative switch
DESCRIPTION:	Left limit state
EXECUTION:	Real time, sampled each **PID** update
CONDITIONAL TO:	**LIMH**, **LIML**, **UCI**, **UCO**, **UCM**
LIMITATIONS:	None
REPORT COMMAND:	**RBm**, **RW** and **RPW** (bit 10)
READ/WRITE:	Read only, set/reset by pin voltage level
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	**0** or **1**
VALUES BY STATE:	**0** = left / negative limit switch not active or pin not assigned as a limit switch **1** = left / negative limit switch active
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

Bm indicates if the Left/Negative pin is presently active. If **Bm** is set to **1**, the historical Left limit flag **Bl** is also set to one. In non PLS firmware motors, the polarity of the signal that is considered active is determined by commands **LIMH** and **LIML**. **[PLS firmware has Active High Limits only!]**

Note on Programmable Software Limits (>=4.76 firmware)
The Active/Real-Time status bit will be set to a one as long real time position is beyond the programmed software limit position.

The Left/Negative Hardware Travel Limit may be disabled by being assigned as a general purpose Input via **UDI** command or Output via **UDO** command. To Re-Enable the Left/Negative Hardware Travel Limit, issue **UDM**.

EXAMPLE:

```
IF Bm
      PRINT("LEFT LIMIT PRESENTLY ACTIVE")
ELSEIF Bl
      PRINT("LEFT LIMIT PREVIOUSLY CONTACTED")
ELSE
      PRINT("LEFT LIMIT NEVER REACHED")
ENDIF
```

Hardware Travel Limit Overview				Status Bits		Command to Clear Historical Bit	Command to Disable Travel Limit Input	Command to Enable Travel Limit Input
Port	Pos/Neg	Plus/Minus	Left/Right	Real Time	Historical			
Port C	Positive	PLUS	RIGHT	**Br**	**Bp**	**Zr, or ZS**	**UCI or UCO**	**UCP**
Port D	Negative	MINUS	LEFT	**Bl**	**Bm**	**Zl, or Zs**	**UDI or UDO**	**UDM**

Related Commands:

BRKTRJ

G

OFF

Z

APPLICATION:	Monitor Motor Off State
DESCRIPTION:	Motor **OFF** state
EXECUTION:	Sampled each **PID** sample
CONDITIONAL TO:	Motor is off
LIMITATIONS:	None
REPORT COMMAND:	**RBo**
READ/WRITE:	Read only. Set by **G**
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	**0** or **1**
VALUE BY STATE:	**1** = Motor is off
	0 = Motor is on
DEFAULT VALUE:	**1**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

Simply stated Bo=0, drive stage is on, Bo =1 dirve stage is off. The Red Drive LED on the motor direclty follows the Bo bit and is therfore a direct indication of the Bo bit. If Bo=1, the Red LED is on. If Bo=0, the Red LED is off.

There are only three reasons that Bo=1.

1. Upon first power-up of a SmartMotor™ and prior to any command that would turn on the drive stage.
2. Any time the OFF command is issued.
3. Any Motor Fault resulting in the OFF command being issued at firmware level.
   a. Position Error (Be=1),
   b. Overheat/RMS-Over-Current (Bh=1),
   c. Exceeding enabled travel limits (Br or Bl detected even briefly).

A motor reset via the **Z** command will also have **Bo** set to one only beacuce it is the same as a Power-up in #1 above.

If **BRKTRJ** has been issued, when a trajectory is not in progress (**Bt** is **0**), the brake is engaged and power is not applied to the motor coils. In this state, **Bo** will not be **0**, even though the amplifier is actually off. This may seem confusing, but it is because the brake is holding the the shaft locked in place nd therefor may be applying a force to the load. **BRKTRJ** is the only mode that behaves this way.

Related Commands:

Bm
Bl
Br
LIMH
LIML
LIMD
LIMN
RS
RPW
RW
UCI
UCP
UCO
UDI
UDM
UDO
SLE
SLD
SLP
SLN
ZI
ZS

APPLICATION: Monitor right limit switch

DESCRIPTION: Right / Positive limit state

EXECUTION: Sampled each **PID** update

CONDITIONAL TO: **LIMH**, **LIML**

LIMITATIONS: None

REPORT COMMAND: **RBp**

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Binary flag

RANGE OF VALUES: **0** or **1**

VALUES BY STATE: **0**= right/positive limit switch not active or pin not assigned as a limit switch

1= right/positive limit switch is active

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

Bp indicates if the right/positive pin is presently active. If **Bp** is set to **1**, the historical right limit flag **Br** is also set to one. In non PLS firmware motors, the polarity of the signal that is considered active is determined by commands **LIMH** and **LIML**. **[PLS firmware has Active High Limits only!]**

Note on Programmable Software Limits (>=4.76 firmware)
The Active/Real-Time status bit will be set to a one as long real time position is beyond the programmed software limit position.

The Right/Positive Hardware Travel Limit may be disabled by being assigned as a general purpose Input via **UCI** command or Output via **UCO** command. To Re-Enable the Right/Positve Hardware Travel Limit, issue **UCP**.

EXAMPLE:

```
IF Br
      PRINT("Right LIMIT PRESENTLY ACTIVE")
ELSEIF Bp
      PRINT("Right LIMIT PREVIOUSLY CONTACTED")
ELSE
      PRINT("Right LIMIT NEVER REACHED")
ENDIF
```

Hardware Travel Limit Overview				Status Bits		Command to Clear Historical Bit	Command to Disable Travel Limit Input	Command to Enable Travel Limit Input
Port	Pos/Neg	Plus/Minus	Left/Right	Real Time	Historical			
Port C	Positive	PLUS	RIGHT	**Br**	**Bp**	**Zr, or ZS**	**UCI or UCO**	**UCP**
Port D	Negative	MINUS	LEFT	**Bl**	**Bm**	**Zl, or Zs**	**UDI or UDO**	**UDM**

Historical Right-Limit Status Bit

Related Commands:

Bm
Bp
Bl
LIMH
LIML
LIMD
LIMN
RS
RPW
RW
UCI
UCP
UCO
UDI
UDM
UDO
SLE
SLD
SLP
SLN
ZI
ZS

APPLICATION:	Monitor Right limit switch latch
DESCRIPTION:	Right limit latch
EXECUTION:	Sampled each **PID** update until latched
CONDITIONAL TO:	**LIMH**, **LIML**
LIMITATIONS:	None
REPORT COMMAND:	**RBr**
READ/WRITE:	Read only. Reset by **RW**, **RS**, **Zr**, **ZS**
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	**0** or **1**
VALUE BY STATE:	**0**= Right/positive limit has not been active
	1= Right /positive limit has been active
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

Br is the historical right limit flag. If the right limit is found to be active during any servo sample, **Br** is set to **1**, and remains **1** until reset by the user. In addition, the motion will stop and the motor will either servo or turn the amplifier **OFF**, depending on the value of the **F** function. The historical right/positive limit flag **Br** provides a latched value in case the limit may have already been contacted (active) but is not at presently active.

The real time Right/Positive limit flag is **Bp**, which only remains set to **1** while the signal level on the user pin C is active. Whenever **Bp** is set to **1, Br** is set to one. The polarity of the signal that is considered active is determined by commands **LIMH** (Active High-To-Stop) and **LIML** (active Low-To_Stop) in all non-PLS firmware motors. PLS firmware motors are always Active High asserted.

If the pin’s function is assigned to being general purpose I/O by use of the **UCI** or **UCO** commands, neither **Bp** nor **Br** will be affected by the pin state. Changing pin states will not elicit limit behavior from the motor. It will be necessary to issue the **UCP** command to assign the pin’s function to being a limt switch, for the pin to again elicit limit behavior, including the setting of **Br**.

(Continued on next page)

Related Commands:

Bm

Bp

Bl

LIMH

LIML

LIMD

LIMN

RS

RPW

RW

UCI

UCP

UCO

UDI

UDM

UDO

SLE

SLD

SLP

SLN

Zl

ZS

In non-PLS firmware motors, Br is reset to zero under the following conditions:

1. When the **S** status byte is accessed for assignment
2. or reported via **RS**, **PRINT(S,#13)** or **PRINT1(S,#13)**
3. or directly reset with **Zr** and **ZS**.
4. or a **G** command is issued AND the **Bp** bit is not set.

In PLS firmware motors, Br is reset to zero under the following conditions:

By issuing either **Zr** and **ZS**.

Example code:

```
IF Br
      PRINT("Right LIMIT PRESENTLY ACTIVE")
ELSEIF Bp
      PRINT("Right LIMIT PREVIOUSLY CONTACTED")
ELSE
      PRINT("Right LIMIT NEVER REACHED")
ENDIF
```

Hardware Travel Limit Overview				Status Bits		Command to Clear Historical Bit	Command to Disable Travel Limit Input	Command to Enable Travel Limit Input
Port	Pos/Neg	Plus/Minus	Left/Right	Real Time	Historical			
Port C	Positive	PLUS	RIGHT	**Br**	**Bp**	**Zr, or ZS**	**UCI or UCO**	**UCP**
Port D	Negative	MINUS	LEFT	**Bl**	**Bm**	**Zl, or Zs**	**UDI or UDO**	**UDM**

Related Commands:

RCS

RCS1

RCKS

RBk

RUN

Z

ZS

APPLICATION:	Monitor Command Syntax Errors
DESCRIPTION:	Command syntax error occurred state
EXECUTION:	Immediate
CONDITIONAL TO:	Syntax error found while executing commands
LIMITATIONS:	None
REPORT COMMAND:	**RBs**
READ/WRITE:	Read only. Reset to zero using **Zs** command
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	**0** or **1**
VALUE BY STATE:	**0**= no syntax error occurred
	1= syntax error detected
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

If a syntax error is encountered in either a serial command or user program, the **Bs** flag is set to **1**. This flag only indicates that a syntax error was encountered. The most common syntax errors are misspellings of commands, but the improper use of variables are also flagged. For example, trying to access the array element **aw[20000]** will also produce a syntax error. If this is the case, the command that contains the syntax error is ignored.

Some errors may appear to be valid syntax, and require other means to detect. To more fully protect against ASCII input stream errors one can use **RCKS**, **RCS**, and **RCS1** commands as well as checking for framing and parity errors.

EXAMPLES:

Suppose host transmitted **A=100** but **A=101** is received due to noise.
Bs would not be set, but **Bb** might be.

Suppose host should have transmitted **A=100** but actually
transmitted **A=L00**.
Bs would be set but **Bb** would not be.

Note: Responses to requests for values in variables or otherwise may cause the Bs bit to be set in any downstream motors on an RS-232 bus or any other motor on a parallel RS-485 bus. The reason for this is because a value (a number) in and of itself is not a valid SmartMotor™ command and as a result, the other motors seeing that response will flag their Bs Bit.

Example:
Issue RP to Motor-1 in a 3 motor system, when Motor-1 responds with it's position in the form of just an integer number, that number in and of itself is not seen as valid command syntax.

Related Commands:

BRKTRJ

G

OFF

S

X

APPLICATION: Monitor Trajectory

DESCRIPTION: Trajectory in progress state flag

EXECUTION: Updated each **PID** sample

CONDITIONAL TO: Trajectory in progress

LIMITATIONS: None

REPORT COMMAND: **RBt**

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Binary flag

RANGE OF VALUES: **0** or **1**

VALUE BY STATE: **0** = no trajectory in progress

1 = trajectory in progress

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The flag **Bt** is set to **1** any time the motor is performing a calculated trajectory path from one point to another. Once the trajectory generator has requested the final target position, the **Bt** flag is reset to zero. At this point, the **PID** positioning control takes over the motion, which means that the motor shaft may still be moving due to mechanical settling.

Torque Mode (MT) will not set the **Bt** bit to **1** because there is no target trajectory.

Mode Velocity (MV) will maintain the Bt bit to 1 regardless of commanded velocity or acceleration even they are set to Zero.

Mode Follow and Mode Step will maintain Bt to 1 even if there are no change in incoming counts.

If a relative or absolute move is commanded in position mode (MP), and there is no (zero) commanded Acceleration or Velocity, the Bt bit will be set to 1 and the motor shaft will not move.

EXAMPLE 1:

```
WHILE Bt     'while trajectory in progress
LOOP
WHILE @V     'while still settling or while velocity not zero
LOOP
OFF          'motor off
BRKENG       'brake engage
```

Related Commands:

BRKTRJ

G

OFF

S

X

EXAMPLE 2:

```
MP              'buffer a position move request
A=10
V=440000
P=10000
G               'start the first buffered move
WHILE Bt        'wait for first trajectory to be done
LOOP            'Note: TWAIT could have been used!
A=20            'buffer another move
V=-222000
P=20000
G               'now begin the second move
```

EXAMPLE 3:

```
MV              'Set to Velocity Mode
A=10
V=440000
G               'start moving
WHILE Bt        'Bt will remain 1 until commanded
LOOP            'otherwise or the motor
                'errors out for some reason
```

Related Commands:

ZS

Zu

APPLICATION: Monitor array index error

DESCRIPTION: Out of range array index state flag

EXECUTION: Latched high upon illegal array access attempted

CONDITIONAL TO: User command attempted to access an array using an illegal index

LIMITATIONS: None

REPORT COMMAND: **RBu**

READ/WRITE: Read only. Reset to zero using **Zu** command

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

VALUE BY STATE: **0** = no illegal array index has occured

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 or higher

DETAILED DESCRIPTION:

The index for each of the **ab[index]**, **aw[index]** and **al[index]** arrays has a valid range. If you go outside the valid range, the system flag **Bu** is set to 1. The syntax error bit **Bs** will also be set to 1. **Bu** is more explicit.

EXAMPLE:

```
Zu                       'reset illegal index flag
t=0
WHILE t<60
   al[t]=t               'initialize array members
   t=t+1                 'to values 0,1,2,3,4….
LOOP
RBu
```

Response is 1 since **al[50]** is the legal end of array.

Related Commands:

Z

G

Bi

RBx

RBi

I

APPLICATION:	Monitor Encoder Wrap Around
DESCRIPTION:	Encoder overflow or underflow occurred
EXECUTION:	Updated each **PID** sample
CONDITIONAL TO:	Position mode set
LIMITATIONS:	Velocity and Torque Modes are immune to encoder wrap around, all others are subject to it.
REPORT COMMAND:	**RBw**
READ/WRITE:	Read only. Reset via **G** or **ZS** command
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Binary flag
RANGE OF VALUES:	**0** or **1**
VALUE OF STATES:	**0**= No encoder wrap around occurred
	1= Encoder wrap around occurred by position mode move
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

If **Bw** is **1**, it indicates that the encoder position has exceeded or "wrapped," beyond maximum value for the 32 bit position register. Specifically, the position has gone outside of the range **-2147483648** to **2147483647**.

This does not at all mean that the SmartMotor™ has lost its position information. It is still tracking its position. If the SmartMotor "wraps" while in Absolute or Relative Position Mode, it will set the Position Error Bit **Be** to **1**, as well.

Velocity mode is designed to survive the wrap around condition and torque mode does not care about any trajectory updates. Neither of these causes **Bw** will set to **1**.

Note: Mode Follow (MF_) allows for a means around wrapping condition by allowing MF0 to be issued on the fly. This will zero out encoder counter registers without having an effect on the motion profile.

Continued on next page.

Related Commands:

Z

G

Bi

RBx

RBi

I

Example to prevent wrap status while in Mode Follow continuously:

```
MF4    'Set to Mode Follow at default 1:1 ratio
WHILE 1
      IF @P>2147480000
            MF0
      ENDIF
      IF @P<-2147480000
            MF0
      ENDIF
LOOP
END
```

Example to prevent wrap status while continuously indexing :

```
UGI                      'Use Port G as general input
D=20000                  'Set relative distance
V=1234567                'Set Velocity
A=123                    'Set Acceleration
WHILE 1      'while forever
      WHILE UGI LOOP     'wait for Port G to be grounded
      G                  'Go (start Moving)
      TWAIT              'Wait until the move is complete
      O=0                'set origin to zero
      WHILE UGI==0 LOOP 'prevent double trigger
LOOP
END
```

Related Commands:

Bi

I

RI

F=

APPLICATION: Monitor Hardware Index Capture Input

DESCRIPTION: Index input state

EXECUTION: Updated each **PID** sample

CONDITIONAL TO: N/A

LIMITATIONS: None

REPORT COMMAND: **RBx**

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Binary flag

RANGE OF VALUES: 0 or 1

VALUE OF STATES: 0 = index capture input is not in contact (low)

1 = index capture input is in contact (high)

DEFAULT VALUE: 0

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Bx is the real-time state of the index input level. The Bx bit is set to a 1 ONLY while the motor is sitting on the index. Be aware that the index marker is only one encoder count wide, this function is mainly used to verify the exact position of the index. For most other uses, it is more efficient to use the functions **Bi** and **I**.

EXAMPLE: (Fast Index Find , Report Bx)

```
MP                 'set buffered velocity mode
A=1000             'set fast acceleration
V=4000000          'set fast velocity
D=2100             'set relative distance just beyond
                   'one shaft turn
i=I                'clear and arm index capture
O=0                'force change to position register
G                  'start fast move
TWAIT              'wait till end of trajectory
P=I                'go back to index
G                  'start motion
TWAIT              'wait until end of trajectory
O=0                'set origin at index

IF Bx
  PRINT("On Index Pulse",#13)
ENDIF
```

BASE

Cam Mode Master Cycle Length

Related Commands:

MC

MC2

MC4

MC8

SIZE

aw[index]

MF1

MF2

MF3

MF4

APPLICATION:	CAM Mode Control
DESCRIPTION:	Cycle period of Mode Cam encoder
EXECUTION:	Immediate
CONDITIONAL TO:	SIZE, MC_, G being issued
LIMITATIONS:	**2** < **BASE** < **32767**
REPORT COMMAND:	N/A
READ/WRITE:	Write only
LANGUAGE ACCESS:	None
UNITS:	Encoder counts
RANGE OF VALUES:	**2** < **BASE** < **32767**
TYPICAL VALUES:	User determined
DEFAULT VALUE:	User determined
FIRMWARE VERSIONS:	4.12 and higher

DETAILED DESCRIPTION:

CAM Mode requires three items to properly perform a cam profile, a **BASE**, **SIZE** and **DATA** table. **BASE** specifies the number of encoder counts that the master turns through one cycle while the slaved, camming SmartMotor™ moves through the points in its data table. **SIZE** is the number of points in the data table.

In the example given below, the camming SmartMotor moves from zero to 120 encoder counts in the positive direction and then back to the zero for every 2000 counts of the master encoder. If the master encoder moves at a constant velocity in the positive direction, this camming profile will continue to repeat for as long as the master encoder continues to move. Since the profile completes every 2000 counts of the master encoder, the **BASE** is **2000**.

The Units are actual encoder counts that are seen at the SmartMotors external encoder input, User ports A and B. This is the same external encoder input that can be read through the counter function **CTR**.

BASE is a parameter required to control Cam Mode motion. In Cam Mode, each value of the external encoder defines a required corresponding SmartMotor position; Cams typically define a periodic motion profile or trajectory. **BASE** defines the number of encoder counts through the external Cam moves before the required position mapping, or required motion, is exactly repeated. Suppose **BASE=10000** encoder counts, and the suppose the required Smart position is to be 100 when the external encoder (**CTR**) reports a value of **2506**, then SmartMotor will be required to be at position 100 whenever **CTR= … -27494**, **-17294**, **2506**, **12506**, **22506, 32506**, etc.

The SmartMotor performs a practical cam application by partitioning the required cam trajectory definition into a number of linearly interpolated segments. The **SIZE** parameter stores the number of segments. The segments are required to partition the **BASE**

Related Commands:

MC

MC2

MC4

MC8

SIZE

Aw[index]

MF1

MF2

MF3

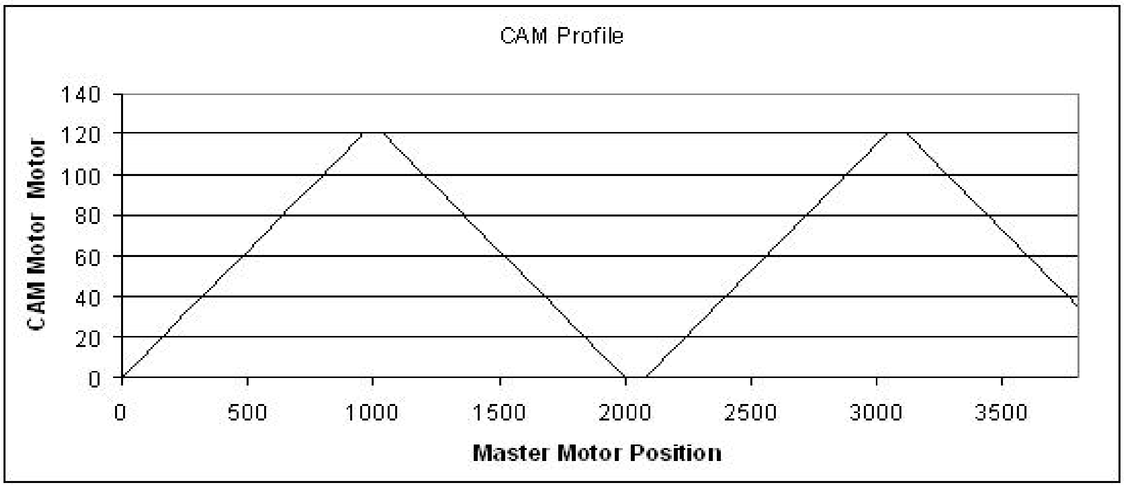
MF4

into a set of equally spaced intervals. Suppose **BASE=1000** and **SIZE=50**. Each segment will then be of width **BASE/SIZE** or 20 counts. The cam motion is then defined by providing the required SmartMotor™ positions corresponding to **CTR= 0, 20, 40, 60 …940, 960** and **980** and **1000**. If the motion is truly periodic the required position at **CTR=0** will identical to the required position at **CTR=1000.**

The cam table is loaded into the **aw[ ]** array, beginning at **aw[0]** and ending with **aw[SIZE]**. It is simplest to define the cam using position at **CTR=0** to be encoder position 0 by issuing **MF0** and **O=0** commands.

EXAMPLE:

A "**saw tooth**" cam with periodic motion every 2000 external encoder counts and the motion interpolation divided into 25 (equal) segments.




```
BASE=2000    'Cam period
SIZE=25      'data segments (number of data points in table)
'CTR data interval = BASE/SIZE = 2000/25 = 80
'CAM motor will be at Data position every 80
'Master encoder counts:
'CTR=0, CTR=80, CTR=160,.... CTR=1840, CTR=1920, CTR=2000
'Now assigning data values beginning with aw[0]:
aw[0] 0 10 20 30 40 50 60 70 80 90 100.
aw[20] 110 120 120 110 100 90 80 70 60.
aw[19] 50 40 30 20 10 0.
MF4    'reset external encoder to zero
O=0    'reset internal encoder position
MC     'buffer CAM Mode
G      'start following the external encoder using cam data
```

The motor will now begin following the External (Master) encoder via the defined CAM profile above.

BRKC

Brake Control Re-Direct to Port C

Related Commands:

BRKENG

BRKRLS

BRKSRV

BRKTRJ

BRKG

BRKI

UCO

APPLICATION:	Hardware brake control
DESCRIPTION:	Re-Direct Brake Control to Port C user Output
EXECUTION:	Immediate and effective until otherwise commanded
CONDITIONAL TO:	BRKI, BRKG
LIMITATIONS:	None
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT STATE:	**BRKI** (Brake Control Default to Internal Bake Pin)
FIRMWARE VERSIONS:	4.15, all PLS firmware. (Not available on 4.40)

DETAILED DESCRIPTION:

SmartMotors™ may be purchased with optional internal zero backlash brakes used to hold a load for safety purposes.

They are Fail Safe Magnetic Clutch Disk Brakes. When power is lost the brake engages. The default with power on is to disengage the brake when ever the drive stage is turned on. The brake takes between 3 and 5 milliseconds to actuate or release.

If an External Brake is used instead of the optional internal brake, the BRKC command allows automatic and interrupt driven control of the external brake via I/O port pin C.

BRKC is a re-direction of the same signal that would otherwise control an internal brake. As a result, Port C will follow the state of the internal brake pin. Port C will be active low (zero volts) when ever the brake should be engaged and at 5VDC when ever the brake should be disengaged.

The logic state follows the present Brake control method chosen.

See **BRKSRV**, **BRKTRJ**, **BRKENG** and **BRKRLS** for more.

Example:

```
UCO         ' Assign Port C to be used as an output pin
BRKC        ' re-direct brake control to port C pin
BRKRLS      ' will set port C to 0VDC
BRKENG      ' will set port C to 5VDC
```

BRKENG

Brake Engage

Related Commands:

BRKRLS

BRKSRV

BRKTRJ

BRKC

BRKG

BRKI

APPLICATION: Hardware brake control

DESCRIPTION: Engages hardware brake immediately

EXECUTION: Immediate and effective until otherwise commanded

CONDITIONAL TO: Hardware **BRAKE** required

LIMITATIONS: None

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT STATE: Power On: **BRKSRV**

Power Off: brake is engaged

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

It is important to turn the servo off when the brake is engaged, or the motor could be driving against the brake and overheat. When the SmartMotor powers up, or comes out of a soft reset, the brake control is set to **BRKSRV** *by default to automatically enforce this safety rule.*

SmartMotors™ may be purchased with optional internal zero backlash brakes used to hold a load for safety purposes.

They are Fail Safe Magnetic Clutch Disk Brakes. When power is lost the brake engages. The default with power on is to disengage the brake when ever the drive stage is turned on. The brake takes between 3 and 5 milliseconds to actuate or release.

When **BRKENG** is issued, the brake is de-energized allowing the magnetic brake to lock the shaft in place.

BRKENG terminates the brake control modes **BRKSRV**, **BRKTRJ**, *and* **BRKRLS.**

NOTE: BRKENG is a manual over-ride to the BRKSRV and BRKTRJ commands. You must subsequently issue either BRKSRV, BRKTRJ, or BRKRLS to allow any further shaft movement !

EXAMPLE:

```
OFF              ' turn motor off
WHILE @V         ' wait for zero velocity
LOOP             ' before
BRKENG           ' applying the brake (shaft locked)
```

Related Commands:

BRKENG

BRKRLS

BRKSRV

BRKTRJ

BRKC

BRKI

UGO

APPLICATION:	Hardware brake control
DESCRIPTION:	Re-Direct Brake Control to Port G user Output
EXECUTION:	Immediate and effective until otherwise commanded
CONDITIONAL TO:	BRKI, BRKC
LIMITATIONS:	None
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT STATE:	**BRKI** (Brake Control Default to Internal Bake Pin)
FIRMWARE VERSIONS:	4.15, all PLS firmware. (Not available on 4.40)

DETAILED DESCRIPTION:

SmartMotors™ may be purchased with optional internal zero backlash brakes used to hold a load for safety purposes.

They are Fail Safe Magnetic Clutch Disk Brakes. When power is lost the brake engages. The default with power on is to disengage the brake when ever the drive stage is turned on. The brake takes between 3 and 5 milliseconds to actuate or release.

If an External Brake is used instead of the optional internal brake, the BRKC command allows automatic and interrupt driven control of the external brake via I/O port pin G.

BRKG is a re-direction of the same signal that would otherwise control an internal brake. As a result, Port G will follow the state of the internal brake pin. Port G will be active low (zero volts) when ever the brake should be engaged and at 5VDC when ever the brake should be disengaged.

The logic state follows the present Brake control method chosen.

See **BRKSRV**, **BRKTRJ**, **BRKENG** and **BRKRLS** for more.

Example:

```
UGO        ' Assign Port G to be used as an output pin
BRKG       ' re-direct brake control to port G pin
BRKRLS     ' will set port G to 0VDC
BRKENG     ' will set port G to 5VDC
```

Brake Control Re-Direct to Port I

Related Commands:

BRKENG

BRKRLS

BRKSRV

BRKTRJ

BRKC

BRKG

APPLICATION: Hardware brake control

DESCRIPTION: Re-Direct Brake Control to Internal Brake Pin

EXECUTION: Immediate and effective until otherwise commanded

CONDITIONAL TO: BRKG, BRKC

LIMITATIONS: None

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT STATE: **BRKI** (Brake Control Default to Internal Bake Pin)

FIRMWARE VERSIONS: 4.15, all PLS firmware. (Not available on 4.40)

DETAILED DESCRIPTION:

SmartMotors™ may be purchased with optional internal zero backlash brakes used to hold a load for safety purposes.

They are Fail Safe Magnetic Clutch Disk Brakes. When power is lost the brake engages. The default with power on is to disengage the brake when ever the drive stage is turned on. The brake takes between 3 and 5 milliseconds to actuate or release.

If an External Brake is used instead of the optional internal brake, the BRKC or BRKG commands allow automatic and interrupt driven control of the external brake via I/O port pin C or G respectively.

BRKI allows the control of the internal brake.

The logic state follows the present Brake control method chosen.

See **BRKSRV**, **BRKTRJ**, **BRKENG** and **BRKRLS** for more.

Example:

```
UGO         ' Assign Port G to be used as an output pin
BRKG        ' Direct brake control to port G pin
BRKI        ' Re-Direct brake control back to internal brake
```

Related Commands:

BRKENG

BRKSRV

BRKTRJ

BRKC

BRKG

BRKI

APPLICATION:	Hardware brake control
DESCRIPTION:	Release hardware break immediately
EXECUTION:	Immediate and effective until otherwise commanded
CONDITIONAL TO:	Hardware **BRAKE** required
LIMITATIONS:	None
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT STATE:	Power on: **BRKSRV**
	Power off: brake engaged
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

It is important to turn the servo off when the brake is engaged, or the motor could be driving against the brake and overheat.

*See **BRKSRV** command.*

SmartMotors™ may be purchased with optional internal zero backlash brakes used to hold a load for safety purposes.

They are Fail Safe Magnetic Clutch Disk Brakes. When power is lost the brake engages. The default with power on is to disengage the brake when ever the drive stage is turned on. The brake takes between 3 and 5 milliseconds to actuate or release.

When **BRKRLS** is issued, the brake is maintained energized allowing full shaft movement.

BRKRLS terminates **BRKSRV** mode, **BRKTRJ** mode, and **BRKENG** condition.

```
BRKENG      ' Assuming motion has stopped
OFF         ' or almost stopped
WAIT=4069
V=0         ' Set buffered velocity
A=0         ' Set buffered acceleration
MP          ' Set buffered mode
P=@P        ' Set Target position to current position
G           ' Begin servo at current position
BRKRLS      ' Release, disengage brake
```

Related Commands:

BRKENG

BRKRLS

BRKTRJ

BRKC

BRKG

BRKI

APPLICATION: Hardware brake control

DESCRIPTION: Release hardware break while motor is on

Engage hardware brake while motor is off

EXECUTION: Immediate and effective until otherwise commanded

CONDITIONAL TO: Hardware **BRAKE** required

LIMITATIONS: None

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT STATE: Power On: **BRKSRV**

Power Off: brake engaged

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

NOTE:

A position error will terminate both the trajectory in progress state and servo on state. In this instance, the brake would then be asserted automatically.

SmartMotors™ may be purchased with optional internal zero backlash brakes used to hold a load for safety purposes.

They are Fail Safe Magnetic Clutch Disk Brakes. When power is lost the brake engages. The default with power on is to disengage the brake when ever the drive stage is turned on. The brake takes between 3 and 5 milliseconds to actuate or release.

It is important to turn the servo off when the brake is engaged, or the motor could be driving against the break and overheat. The **BRKSRV** command does this for you by releasing the brake automatically whenever the motor is on and engaging it whenever the motor turns off for any reason. Another way of looking at this is, the brake will be applied whenever the motor off bit **Bo** is **1**.

BRKSRV terminates the brake control modes **BRKENG**, **BRKTRJ**, and **BRKRLS**.

```
BRKSRV      'set brake mode assuming it is safe
MP          'set buffered mode
A=100       'set buffered acceleration
V=100000    'set buffered maximum velocity
P=1000      'set target
G           'servo on, brake release, go to target
```

BRKTRJ

Brake Engage With No Active Trajectory

Related Commands:

BRKENG

BRKRLS

BRKSRV

BRKC

BRKG

BRKI

APPLICATION:	Hardware brake control	
DESCRIPTION:	Release hardware brake while a trajectory is in progress Engage brake, turn off servo while no trajectory is in progress	
EXECUTION:	Immediate and effective until otherwise commanded	
CONDITIONAL TO:	Hardware **BRAKE** required	
LIMITATIONS:	None	
REPORT COMMAND:	N/A	
READ/WRITE:	N/A	
LANGUAGE ACCESS:	N/A	
UNITS:	N/A	
RANGE OF VALUES:	N/A	
TYPICAL VALUES:	N/A	
DEFAULT STATE:	Power On:	**BRKSRV**
	Power Off:	brake engaged
FIRMWARE VERSIONS:	4.00 and higher	

DETAILED DESCRIPTION:

SmartMotors™ may be purchased with optional internal zero backlash brakes used to hold a load for safety purposes.

They are Fail Safe Magnetic Clutch Disk Brakes. When power is lost the brake engages. The default with power on is to disengage the brake when ever the drive stage is turned on. The brake takes between 3 and 5 milliseconds to actuate or release.

BRKTRJ automatically coordinates movement and brake application. When a trajectory is started by a **G** command, the brake is released. When the trajectory completes the brake is engaged and, simultaneously, the servo is turned off. In this mode, and whenever the motor is not performing a trajectory, the brake is automatically engaged and the servo turned off for any reason that the Bt (Busy Trajectory Bit) clears.

A consequence of this behavior is that any non-trajectory mode, like torque mode, will not result in motion, as the brake will be engaged and the servo will be off. This could be confusing to a user unaware of the nature of **BRKTRJ**, especially since the motor-off flag **Bo** is **0** or false. To understand this, from an operating control mode point of view, the motor has not changed modes to **OFF**, which would be coincidental with **Bo** set to **1**. When running in torque or some other non-trajectory mode, it is more appropriate to use **BRKSRV**

BRKTRJ terminates the **BRKSRV** mode, **BRKENG** condition, and **BRKRLS** condition.

Brake Engage With No Active Trajectory

Related Commands:

BRKENG

BRKRLS

BRKSRV

BRKC

BRKG

BRKI

One consequence of **BRKTRJ** is that the trajectory flag is reset to zero immediately when trajectory generator declares the trajectory to be over. At this instant, the **BRKTRJ** will engage the brake (de-energize the brake)

```
      BRKTRJ      'set brake mode to follow Bt bit.
      MP          'set buffered mode
      A=100       'set buffered acceleration
      V=100000    'set buffered maximum velocity
      C1          'program statement label
      P=1000      'set buffered target position
      G           'servo on, start trajectory
(The brake will automatically be energized and released)
      TWAIT       'wait for trajectory to end
                  'now brake will be on and servo off
      WAIT=4069   'brake on for ~one second
      P=0         'set new buffered target position
      G           'servo on, brake off, trajectory
      WAIT=4069
        GOTO1     'effective loop forever
```

Note: A position error will terminate the trajectory in progress state. In this case, brake would then be asserted.

Once in BRKTRJ mode, the brake can be audibly hear clicking on at the beginning of each move and clicking back off at the end of each move.

This is normal and gives assurance of proper operation.

BREAK

Program Flow Loop Exit Control

Related Commands:

CASE

DEFAULT

ENDS

LOOP

SWITCH

WHILE

APPLICATION:	Program execution flow control
DESCRIPTION:	Causes immediate exit from a **WHILE** or **SWITCH** control block
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Downloaded code only, not via Serial Port !
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

BREAK is used both by **WHILE . . LOOP** and **SWITCH . . ENDS** control flow structure blocks. In both structures, if **BREAK** is encountered the program jumps out of that particular **WHILE** loop or **SWITCH** structure. If the control blocks are to be nested, **BREAK** only exits the **WHILE** loop or **SWITCH** structure that it is currently in.

The most common use of **BREAK** is to end each **CASE** of a **SWITCH** control structure. Without the **BREAK** statement, the program would continue to execute into the next **CASE**, even if it is not true.

EXAMPLE:

```
SWITCH a
      CASE 1
            PRINT("Hiya!",#13)
      CASE 2
            PRINT("Lo there!",#13)
      BREAK
      CASE 3
            PRINT("Me here!",#13)
      BREAK
      DEFAULT
            PRINT("Urp!",#13)
      BREAK

ENDS
```

If **a=2**, the SmartMotor™ will print "Lo there!" If **a=1**, however, the SmartMotor will print both "Hiya!" and "Lo there!" There is no **BREAK** statement to stop the program from running into case 2.

Related Commands:

CASE

DEFAULT

ENDS

LOOP

SWITCH

WHILE

BREAK could always be replaced by **GOTO**, and this is how it is actually executed using the precompiled program location. **BREAK** has the advantages of not requiring a statement label to define the program branch location and conforming to structured programming methodology.

BREAK is not a valid terminal command, it is only valid from within a user program. If you want to be able to "break out of" a control block by remote (terminal) commands you will need to use **GOTO#** or **GOSUB#** and appropriate statement labels. The example illustrates this concept.

EXAMPLE:

```
 a=1
 WHILE a
    PRINT("I am still here …",#13)
       WAIT=12000
    IF a==100
        BREAK       'a=100 could be sent via serial command
     ENDIF
 LOOP
 GOTO20
 C10
 PRINT("EXITED with a==100",#13)
 END
 C20
 PRINT("EXITED with a<0",#13)
END
```

C{statement_label_number}

Program Subroutine Label

Related Commands:

GOSUBnnn

GOTOnnn

APPLICATION: Program execution flow control

DESCRIPTION: Program statement label

EXECUTION: N/A

CONDITIONAL TO: N/A

LIMITATIONS: Pre 4.00 firmware only permits labels **C0…C9**

Firmware 4.00 and higher permits labels **C0…C999**

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

C{number} is a statement label, where "number" is a value between **0** and **999**. Statement labels mainly provide the internal addressing required to support the **GOSUB{number}** and **GOTO{number}** language commands. For example **GOTO1** directs the program to label **C1**, while **GOSUB37** directs the program to the subroutine that starts at label **C37**. You can also use labels to simply enhance program clarity. Statement labels may be placed anywhere within a program except in the middle of an expressions.

The program labels work via a jump table in the header of the compiled program. The header contains the location of every label from **0** up to the highest label value used.

EXAMPLES: (consider these two programs)

```
    C0
    END
and
    C999
    END
```

The first compiled program (**C0 . . END**) will be much smaller than the second (**C999 . . END**), even though they behave exactly the same.

The program header is read whenever the SmartMotor™ powers up or is reset. This means that the SmartMotor knows how to jump to any label location, even if the

Related Commands:

GOSUBnnn

GOTOnnn

program has never been run, and start executing the program from there. This is a common means of making a single program that contains several routines that can be invoked on demand from a host.

EXAMPLE:

```
END
C0
    PRINT("Routine 0",#13)
    END
C1
    PRINT("Routine 1",#13)
    END
C2
    PRINT("Routine 2",#13)

END
```

To run routine **1**, the host simply issues **GOTO1** to the SmartMotor™. If the host issues **GOTO3**, routine 3 is run. You can use a similar technique to allow the host to control where the program starts.

Using **GOTOnnn** to jump to a location within a **SWITCH** block may be syntactically valid but yield unexpected runtime program execution when **CASE** number is encountered.

It is also possible to use **IF**, **WHILE**, and **SWITCH** to provide such multiple choice program start points.

EXAMPLES:

```
IF a==6
    C0
    G
ENDIF

GOTO5                           'valid syntax
SWITCH a
    CASE 1   PRINT("1")
C5  CASE 2   PRINT("2")         'at runtime "2" will be
ENDS                            'transmitted END
```

CCHN(type,channel)

Close Communications Channel

Related Commands:

OCHN( )

Z

APPLICATION:	Communications control
DESCRIPTION:	Close a communications channel
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
PARAMETERS:	Type= **RS2**, **RS4**
	Channel = **0** or **1**
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

CCHN(type,channel) closes the specified communications channel, where "type" is the communications mode, and "channel" is the comm port you want to close. This command flushes the serial port buffer and any characters still in the buffer will be lost. The channel **0** comm port can only be RS-232 or RS-485, while channel **1** can only be RS-485.

Valid **CCHN** commands:

```
CCHN(RS2,0) 'Close the channel 0 RS232 port

CCHN(RS4,1) 'Close the channel 1 RS485 port
```

After power up or **Z** reset command, channel **0** is opened

as RS232 by default.

Related Commands:

Bb

Bc

Bf

Bs

CHN0

CHN1

Zs

APPLICATION:	Serial communications control
DESCRIPTION:	Fetch combined serial communications error event flags
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Cannot assign value of **CHN**
REPORT COMMAND:	**RCHN**
READ/WRITE:	Report value only
LANGUAGE ACCESS:	Report via **RCHN** only
UNITS:	Set of 4 binary state flags
PARAMETERS:	Type= **RS2**, **RS4**, or **IIC**
	Channel = **0** or **1**
RANGE OF VALUES:	**0** to **15**
TYPICAL VALUES:	**0**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

The read only function **CHN** holds binary coded historical error information about the two serial channels on the SmartMotor™. It gives the 4 bit status of either serial port channels **0** or **1**, broken down as follows:

CHN bit 0= 1 if either receive buffer has overflowed

CHN bit 1= 1 if a framing error occurred on either channel

CHN bit 2= 1 if a scan error occurred on either channel

CHN bit 3= 1 if a parity error occurred on either channel

For example, if **RCHN** returns a 4, it means that a scan error was detected on channel **0** or channel **1**. You cannot tell, however, whether the syntax error was on channel **0**, **1** or both. If you really must know, you would issue **RCHN0** and **RCHN1**, which return the 4 bit status of the individual serial ports.

CHN is read only, but cannot be assigned to a variable. It can be reported through **RCHN**, **PRINT(CHN,#13)** and **PRINT1(CHN,#13)** as well.

Related Commands:

Bb

Bc

Bf

Bs

CHN0

CHN1

Zs

Each of the four bits of **CHN** correspond to one of the four communications system status bytes:

Bc= CHN bit 0

Bf= CHN bit 1

Bs= CHN bit 2 AND User Program Scan Error

Bb= CHN bit 3

Related Commands:

CHN

CHN1

RCHN

RCHN0

RCHN1

APPLICATION: Serial communications control

DESCRIPTION: Fetch serial communications channel 0 error event flags

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **RCHN0**

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Set of 4 binary state bits

RANGE OF VALUES: **0** to **15**

TYPICAL VALUES: **0**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

CHN0 holds binary coded historical error information regarding the channel **0** communications channel. It gives the 4 bit status of the primary, or channel **0**, serial port, broken down as follows:

> **CHN0 bit 0= 1** if the primary receive buffer has overflowed
>
> **CHN0 bit 1 = 1** if a framing error occurred on channel 0
>
> **CHN0 bit 2= 1** if a scan error occurred on channel 0
>
> **CHN0 bit 3= 1** if a parity error occurred on channel 0

If **RCHN0** returns a **4**, it means that a scan error was detected on channel 0. If **CHN0** equals zero, no error has been detected since opening the channel.

CHN0 is read only, but cannot be assigned to a variable. It can be reported through **RCHN0**, as already seen, and **PRINT(CHN0,#13)** and **PRINT1(CHN0,#13)** as well.

SEE EXAMPLES ON FOLLOWING PAGE:

Related Commands:

CHN

CHN1

RCHN

RCHN0

RCHN1

EXAMPLE:

The host transmitted **A=100** but the serial port actually received **K=100** then tried to execute **K=100**

```
PRINT(CHN0)     'responds to host with 4

                'since K= is invalid
```

EXAMPLE: (test individual flags)

```
IF CHN0&4
      PRINT("HOST CHANNEL - scan error occurred")
ELSEIF CHN0&1
      PRINT("HOST CHANNEL - buffer overflow")
ENDIF
```

EXAMPLE: (test all flags)

```
IF CHN0
      PRINT("SERIAL ERROR !!")
ENDIF
```

Related Commands:

CHN

CHN0

RCHN

RCHN0

RCHN1

APPLICATION: Serial communications control

DESCRIPTION: Fetch serial communications channel 1 error event flags

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **RCHN1**

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Set of 4 binary state bits

RANGE OF VALUES: **0** to **15**

TYPICAL VALUES: **0**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

CHN1 holds binary coded historical error information regarding the channel 1 communications channel. It gives the 4 bit status of the channel 1 serial port, broken down as follows:

CHN1 bit 0= 1 if the primary receive buffer has overflow

CHN1 bit 1= 1 if a framing error occurred on channel 0

CHN1 bit 2= 1 if a scan error occurred on channel 0

CHN1 bit 3= 1 if a parity error occurred on channel 0

If **RCHN1** returns a **4**, it means that a scan error was detected on channel 1. If **CHN1** equals zero, no error has been detected since opening the channel.

CHN1 is read only, but cannot be assigned to a variable. It can be reported through **RCHN1**, as already seen, and **PRINT(CHN1,#13)** and **PRINT1(CHN1,#13)** as well.

SEE EXAMPLES ON FOLLOWING PAGE

Related Commands:

CHN

CHN0

RCHN

RCHN0

RCHN1

EXAMPLE:

Host transmitted **A=100** but the serial port actually received **K=100** then tried to execute **K=100**

```
PRINT(CHN1)        'responds to host with 4
                   'since K= is invalid
```

EXAMPLE: (test individual flags)

```
IF CHN1&4
      PRINT("CHANNEL 1 - scan error occurred")
ELSEIF CHN1&1
      PRINT("CHANNEL 1 - buffer overflow")
ENDIF
```

EXAMPLE: (test all flags)

```
IF CHN1
          PRINT("CHANNEL 1 SERIAL ERROR !!")
ENDIF
```

Related Commands:

RCLK

WAIT

APPLICATION:	Hardware clock access
DESCRIPTION:	Value of free running firmware clock
EXECUTION:	Incremented once each PID sample
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	**RCLK**
READ/WRITE:	Read only
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Number
RANGE OF VALUES:	**0** to **2147483647**
TYPICAL VALUES:	Sequential
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

CLK is an independent, free running, read write counter. It is reset to zero upon a hardware or software reset, and it increments once per **PID** cycle. The default **PID** rate produces ~4069 samples per second, so there are roughly four **CLK** ticks per millisecond at **PID1**. If the **PID** sample is modified by **PID2**, **PID4** or **PID8**, the amount of time associated with one **CLK** tick will increase by 2x, 4x or 8x, respectively. The user may also assign a value to this counter at any time. **CLK** is 31 bits in size and will roll over (return to zero) at value **2,147,483,647**, which corresponds to 4.13 days at **PID1**.

EXAMPLE 1:

The following two examples perform the same function, pause for one second:

```
WAIT=4069                'Pause for one sec
CLK=0                    'Initialize clock
WHILE CLK<4069           'Loop one sec

LOOP
```

Related Commands:

RCLK

WAIT

The advantage of the second example is that you could write code within the **WHILE** loop to execute during the pause.

EXAMPLE 2:

CLK increments more slowly at **PID2** than **PID1** etc.

To most easily see the effect, load and run the following code.

```
PID1
a=5
WHILE a
      a=a-1
      CLK=20
      WHILE CLK<4089 LOOP 'note nested whiles are permitted
      PRINT("PID1",#13)
LOOP
a=5
PID2
WHILE a
      a=a-1
      CLK=20
      WHILE CLK<4089 LOOP
      PRINT("PID2",#13)
LOOP
PID4
a=5
WHILE a
      a=a-1
      CLK=20
      WHILE CLK<4089 LOOP
      PRINT("PID4",#13)
LOOP
PID1                          'return to PID1
END
```

CMD

Accept Command Input RS-232

Related Commands:

CMD1

DAT

DAT1

OCHN

APPLICATION:	Serial communications control Parameter
DESCRIPTION:	Set serial communication channel 0 to receive commands
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT STATE:	Command channel
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

By default, anything received over the primary serial port is interpreted as a command. By configuration, however, both the primary and channel 1 serial ports can treat incoming information as either commands or data. The **CMD** function tells the SmartMotor™ to interpret information coming into the primary port as standard commands.

The alternate to **CMD** is **DAT**, which causes the SmartMotor to simply store incoming bytes in the 16 character serial buffer. The characters are read from the buffer with the **GETCHR** command, while the **LEN** function holds the number of characters in the buffer.

WARINING !! Issuing DAT at the command line will prevent the motor from responding to any further commands via Com 0 (RS-232 Port) and will essentially lock you out of the motor !!!

It is a good idea to devise a means of invoking **CMD** via I/O or specific serial data if you use data mode.

See next Page for Examples.

Related Commands:

CMD1

DAT

DAT1

OCHN

EXAMPLE: (using the default host channel)

```
PRINT(#13,"Default mode is CMD")
PRINT(#13,"Issuing DAT")
DAT
PRINT(#13,"Issuing a=GETCHR")
PRINT(#13,"Use SMI to send RP command",#13)
a=GETCHR
b=GETCHR
c=GETCHR
PRINT(#13,"Received ASCII ",a)
PRINT(#13,"Received ASCII ",b)
PRINT(#13,"Received ASCII ",c)
PRINT(#13,"Issuing CMD")
CMD
IF a==82 GOTO10 ENDIF   'validate user command
IF b==80 GOTO10 ENDIF   'sent via SMI
IF c==32 GOTO10 ENDIF
PRINT(#13,"Use SMI to send RP command")
PRINT(#13,"You should see a motor response",#13)
END
C10
PRINT(#13,"PROGRAM DID NOT RECEIVE RP COMMAND")
PRINT(#13,"PROGRAM ABORTING",#13)
END
```

CMD1

Accept Command Input RS-485

Related Commands:

CMD

DAT

DAT1

OCHN

APPLICATION: Serial communications control

DESCRIPTION: Set serial communication channel 1 to receive commands

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT STATE: Command channel

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

By default, anything received over the secondary serial port is interpreted as a command. By configuration, however, channel 1 serial port can treat incoming information as either commands or data. The **CMD1** function tells the SmartMotor™ to interpret information coming into the channel 1 port as commands.

The alternate to **CMD1** is DAT1, which causes the SmartMotor to simply store incoming bytes in the 16 character serial buffer. The characters are read from the buffer with the **GETCHR1** command, while the **LEN1** function holds the number of characters in the buffer. For details about the use of data mode, please refer to the **DAT1** command.

WARINING !! Issuing DAT1 at the command line will prevent the motor from responding to any further commands via Com 1 (RS-485 Port) and will essentially lock you out of the motor !!!

It is a good idea to devise a means of invoking **CMD1** via I/O or specific serial data if you use data mode.

See next page for example:

Related Commands:

CMD

DAT

DAT1

OCHN

EXAMPLE: (using the default channel 1)

```
PRINT1(#13,"Default mode is CMD")
PRINT1(#13,"Issuing DAT")
DAT
PRINT1(#13,"Issuing a=GETCHR")
PRINT1(#13,"Use SMI to send RP command",#13)
a=GETCHR
b=GETCHR
c=GETCHR
PRINT1(#13,"Received ASCII ",a)
PRINT1(#13,"Received ASCII ",b)
PRINT1(#13,"Received ASCII ",c)
PRINT1(#13,"Issuing CMD")
CMD1
IF a==82 GOTO10 ENDIF   'validate user command
IF b==80 GOTO10 ENDIF   'sent via SMI
IF c==32 GOTO10 ENDIF
PRINT1(#13,"Use SMI to send RP command")
PRINT1(#13,"You should see a motor response",#13)
END
C10
PRINT1(#13,"PROGRAM DID NOT RECEIVE RP COMMAND")
PRINT1(#13,"PROGRAM ABORTING",#13)
END
```

Related Commands:

ENC0

ENC1

MC

MF

MF0

MF1

MF2

MF4

MFR

MS

MS0

MSR

** Some low cost SmartMotors™ do not have second encoder input capability.*

APPLICATION:	External Encoder
DESCRIPTION:	External encoder counter reading
EXECUTION:	Updated once each **PID** sample
CONDITIONAL TO:	External encoder input signal available
	ENC0 and **ENC1** commands - see example below
LIMITATIONS:	None
REPORT COMMAND:	**RCTR**
READ/WRITE:	Read only
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**0**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

By Default, **CTR** contains the present value for the secondary encoder (or Step and Direction) signals. **ENC0** and **ENC1** determine whether the internal or external inputs are primary or secondary. **ENC0** is the default state. This means that the internal encoder will be the primary encoder and Ports A and B will be the source for Phase A and B (or Step and Direction) of an external source. Under this condition, **CTR** will contain the position or count value for Ports A and B. Unlike using **O=***expression* for the internal encoder counter, **CTR** cannot be set to any specific value. It can only be set to zero

If you issue **MS0, MF0**, **MF1**, **MF2**, or **MF4**, **CTR** will be set to zero and Ports A and B will be set to receive phase A and B of a standard quadrature encoder. If the external encoder changes position. **RCTR** will report that value.

If you issue **ENC1, CTR** will be set to zero and the sources of **CTR** and **@P** will swap. Now **CTR** will reflect internal encoder position and **@P** will reflect external encoder position.

If you issue **ENC0**, the sources will swap back to default and again **CTR** will follow the external encoder.

MF0 and **MS0** will both set **CTR** to Zero without changing the mode of operation.

(Continued on next page)

Related Commands:

ENC0

ENC1

MC

MF

MF0

MF1

MF2

MF4

MFR

MS

MS0

MSR

EXAMPLE:

To better understand the meaning of **CTR**; try the following with a SmartMotor™.

```
      O=1234            'Set origin to zero
Then issue:
      RP                'response will be 1234
Then issue:
      ENC1              'make INTERNAL encoder the source of
CTR
Then issue:
      RP                'response will be zero
      RCTR              'response is also zero 0
                        'Physically turn the motor shaft and
                        'Query the position again
      RP                'response should again be that
                        'NON ZERO response obtained before
      RCTR              'response is another non zero number
      ENC0              'return internal motor shaft encoder to
                        'Normal functioning
```

If you have an external encoder, attach it to a SmartMotor and repeat the above sequence or some similar sequence.

If in gear mode (Mode Follow via MF(n)) and you issue MF0 on the fly, CTR will be set to zero while trajectory continues without any glitch in movement. This serves two purposes. One, it gives a means to zero the counter while moving. Two, it allows the user to prevent Wrap status from occurring should CTR exceed +/-2^31.

Related Commands:

A

G

MP

MF1

MFR

P

V

APPLICATION: Trajectory control

DESCRIPTION: Relative move distance for position mode

EXECUTION: Buffered pending a **G** command

CONDITIONAL TO: Position mode. See **MFR** command for alternate usage.

LIMITATIONS: Encoder wrap around will produce a position error

REPORT COMMAND: **RD**

READ/WRITE: Read write

LANGUAGE ACCESS: Assignment, expressions and conditional testing

UNITS: Encoder counts

RANGE OF VALUES: **-2147483648** to **2147483647**

TYPICAL VALUES: N/A

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The D command can be used during gearing to implement Dynamic Phase Adjust

(See MFR).

The D command can also be sued in CAM mode to implement a dwell between CAM cycles.

D=expression commands a relative distance move from the present position and will be repeated every time a **G** command is issued. It is a signed value allowing a relative move in either direction.

If you command a **D** move while the motor shaft is moving, its starting point will be the actual shaft position when the **G** command is executed. In other words, the **D** move will be relative to the reception of the **G** command on-the-fly. This method will result in accumulating drift.

To avoid drift, If you issue the command **D=100** and then enter the **G** command ten times each **after the previous move has completed**, you will travel a total of precisely 1000 counts regardless of any following error at the end of the previous moves. The **D** move starts from where you are supposed to be, regardless of the present position error, avoiding the problem of position drift or accumulating errors over several relative moves.

In downloaded code, you would use the **TWAIT** command prior to the next **G** command. In doing so, the next **G** will not be issued until the previous trajectory has completed.

Relative Moves are subject to wrap status. If the next relative move causes the counter to exceed +/- 2^31 counts, the motor will error out. The following code example will allow continuous indexing without exceeding maximum count.

Continued on next page

Related Commands:

P

A

V

G

MP

MF1

MFR

Example

(Continuous Index Moves with no accumulated error or roll over)

```
O=0                             'reset origin          A=100
                                'Set Acceleration      V=100000
                                'Set Velocity          D=20000

                                'Set Relative distance
MP
                                'Set to Position Mode
WHILE 1                         'While Forever.......

      G
                                'Initiate Index Move
      TWAIT                     'Wait until Move is Completed

      O=0                       'Reset Position to Zero
LOOP                            'loop back to repeat continuously
END
```

In the above example, the motor counts will continuously increase to 20000 during each move and then be set back to zero at the end of each move. There will be no accumulating error because the **O=(expression)** command accounts for any following error that may be present after the trajectory has completed.

The D command is also used during gearing to implement Dynamic Phase Adjust

(See MFR).

Phase Offset Moves using the D command.

While in gearing (Mode Follow or Step Mode), the motor will follow an external encoder or pulse and direction signal. The D command allows a move within gearing to adjust the shaft position forward or backwards .

Suppose the motor is set on Mode follow and is following a conveyor at a continuous speed of 1000RPM. If the shaft needs to be moved forward 2000 counts, you can enter **D=2000**, **V=**(speed relative to machine base), and **G** and the motor will move forward in it's gearing trajectory by 2000 counts.

This method may be used for printing alignment on electronic line shafts. It may also be used for tension control between two motors feeding a product through nip rollers. Phase offset moves allow for anti-backlash where two motors drive the same gear or load from the same point. It may also be used for adjustment and alignment of wide gantries where there may be two X or two Y motors.

Related Commands:

CMD

CMD1

DAT1

APPLICATION: Serial communications control

DESCRIPTION: Set serial communication channel 0 to receive data

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: Applies to Com Channel 0 (main RS-232 Port)

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT STATE: Command channel (See **CMD**)

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

By default, anything received over the primary serial port is interpreted as a command. By configuration, however, incoming information can be parsed as general data instead of actual command data. The **DAT** applies to the primary Com channel 0 port and will simply store incoming bytes in the 16 character serial buffer without attempting to execute any of that data. The characters are read from the buffer with the **GETCHR** command, while the **LEN** function holds the number of characters in the buffer. With proper code writing a custom serial command parser can be created.

Warning: The **DAT** command should only be used within the context of a downloaded program with proper code to follow that deals with all incoming serial data from that point on. If **DAT** is issued via serial port, you will be immediately locked out of the motor until next power-up. It is highly recommended to write code that will handle any incoming data and allow a means to issue **CMD** command within that code to re-open standard command mode via serial port.

The following code example is written to parse out incoming data. It specifically looks for the characters R, P, and (space key) one by one. Each incoming character is stored into 3 consecutive variables. Then they are compared to the proper ASCII value to insure they match. If the match, the program prints acknowledgment of it.

SEE NEXT PAGE FOR CODE EXAMPLE

Related Commands:

CMD

CMD1

DAT1

LEN

OCHN

EXAMPLE: (using the default host channel)

```
PRINT(#13,"Default mode is CMD")
PRINT(#13,"Issuing DAT")
DAT
PRINT(#13,"Issuing a=GETCHR")
PRINT(#13,"Use SMI to send RP command",#13)
a=GETCHR
b=GETCHR
c=GETCHR
PRINT(#13,"Received ASCII ",a)
PRINT(#13,"Received ASCII ",b)
PRINT(#13,"Received ASCII ",c)
PRINT(#13,"Issuing CMD")
CMD
IF a!=82 GOTO10 ENDIF   'check for "R"
IF b!=80 GOTO10 ENDIF   'check for "P"
IF c!=32 GOTO10 ENDIF   'check for space character
PRINT(#13,"Use SMI to send RP command")
PRINT(#13,"You should see a motor response",#13)
END
C10
PRINT(#13,"PROGRAM DID NOT RECEIVE RP COMMAND")
PRINT(#13,"PROGRAM ABORTING",#13)
```

Related Commands:

CMD

CMD1

DAT

APPLICATION:	Serial communications control
DESCRIPTION:	Set serial communication channel 1 to receive data
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Applies to Com Channel 1 (Alternate RS-485 Port)
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT STATE:	Command channel
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

By default, anything received over the secondary serial port is interpreted as a command. By configuration, however, incoming information can be parsed as general data instead of actual command data. The **DAT1** applies to the secondary Com channel 1 port and will simply store incoming bytes in the 16 character serial buffer without attempting to execute any of that data. The characters are read from the buffer with the **GETCHR1** command, while the **LEN1** function holds the number of characters in the buffer. With proper code writing a custom serial command parser can be created.

Warning: The **DAT1** command should only be used within the context of a downloaded program with proper code to follow that deals with all incoming serial data from that point on. If **DAT1** is issued via serial port, you will be immediately locked out of the motor until next power-up. It is highly recommended to write code that will handle any incoming data and allow a means to issue **CMD1** command within that code to re-open standard command mode via serial port.

The following code example is written to parse out incoming data. It specifically looks for the characters R, P, and (space key) one by one. Each incoming character is stored into 3 consecutive variables. Then they are compared to the proper ASCII value to insure they match. If the match, the program prints acknowledgment of it.

SEE NEXT PAGE FOR CODE EXAMPLE

Related Commands:

CMD

CMD1

DAT1

LEN

OCHN

EXAMPLE: (using the secondary com channel 1)

```
PRINT1(#13,"Default mode is CMD1")
PRINT1(#13,"Issuing DAT1")
DAT1
PRINT1(#13,"Issuing a=GETCHR1")
PRINT1(#13,"Use SMI to send RP command",#13)
a=GETCHR1
b=GETCHR1
c=GETCHR1
PRINT1(#13,"Received ASCII ",a)
PRINT1(#13,"Received ASCII ",b)
PRINT1(#13,"Received ASCII ",c)
PRINT1(#13,"Issuing CMD1")
CMD1
IF a!=82 GOTO10 ENDIF   'check for "R"
IF b!=80 GOTO10 ENDIF   'check for "P"
IF c!=32 GOTO10 ENDIF   'check for space character
PRINT1(#13,"Use SMI to send RP command")
PRINT1(#13,"You should see a motor response",#13)
END
C10
PRINT1(#13,"PROGRAM DID NOT RECEIVE RP COMMAND")
PRINT1(#13,"PROGRAM ABORTING",#13)
END
```

DEFAULT

Switch-Case Structure Element

Related Commands:

BREAK

CASE

ENDS

SWITCH

APPLICATION:	Program execution control
DESCRIPTION:	Default for **SWITCH** program control block
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Must reside within a **SWITCH** and **ENDS** structure
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

DEFAULT allows controlled code execution in a **SWITCH** structure for non-**CASE** evaluated results. In the following example, DEFAULT is used when no VASE can be executed for the value of "x".

EXAMPLE 1:

```
SWITCH x
      CASE 1
            PRINT("x=1",#13)
      BREAK
      CASE 2
            PRINT("x=2",#13)
      BREAK
      CASE 3
            PRINT("x=3",#13)
      BREAK
      DEFAULT
            PRINT("x does not equal 1, 2 or 3,#13)
      BREAK

ENDS
```

The first line, **SWITCH x**, lets the SmartMotor™ know that it is checking the value of the variable x. The second line, **CASE 1**:, begins the section of code that tells the SmartMotor what to do if x is equal to **1**. Similarly, the 8th line, **CASE 3**:, tells what to do if x**=3**. Finally, **DEFAULT**, tells what to do if none of the **CASE'**s match the value of the x.

Related Commands:

BREAK

CASE

ENDS

SWITCH

If no **CASE** number equals the value of the **SWITCH** expression and there is no **DEFAULT** case, program execution passes through the **SWITCH** control block to the **ENDS** statement without explicitly performing any commands.

There can only be one **DEFAULT** statement per **SWITCH** control block.

DEFAULT is not a valid terminal command, it is only valid within a user program.

EXAMPLE 2:

```
a=20
WHILE a
      SWITCH a-12
            CASE -4 PRINT("-4 ") BREAK
            CASE -3 PRINT("-3 ") BREAK
            CASE -2 PRINT("-2 ") BREAK
            CASE -1 PRINT("-1 ") BREAK
            CASE 0 BREAK
            CASE 1 PRINT("+1 ") BREAK
            CASE 2 PRINT("+2 ") BREAK
            CASE 3 PRINT("+3 ") BREAK
            CASE 4 PRINT("+4 ") BREAK
            DEFAULT PRINT("D ")
      ENDS
a=a-1

LOOP
```

The above code example produces the following output:

D D D D +4 +3 +2 +1 -1 -2 -3 -4 D D D D D D D

DIN{port}{channel}

Input Byte From I/O Device

Related Commands:

DOUT

APPLICATION:	Input control
DESCRIPTION:	Fetch AniLink digital peripheral input byte
EXECUTION:	Immediate byte read from IIC link
CONDITIONAL TO:	Peripheral input attached to motor
LIMITATIONS:	Port= A . . H and Channel= 0 . . 63
REPORT COMMAND:	**RDIN{Port}{channel}**
READ/WRITE:	Read only
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Number
RANGE OF VALUES:	**0** to **255**
TYPICAL VALUES:	**0** to **255**
DEFAULT VALUE:	**255**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The **DIN{Address}{Channel}** is used to read the single byte integer value of a given address and channel from a peripheral I/O device such as the DIO-100 or OPTO-1 digital I/O expansion module. The value is received via the AniLink communications channel. The "address" parameter must correspond with hardware address jumpers on the peripheral expansion card. The Addresses are designated as A, B, C, D, E, F, G, or H. The "channel" number, which may be from 0 to 63, is device specific. Typically it is 0 thru 8. See the specific peripheral user manual for specific details.

See Appendix ? for greater detail and information about expanding the SmartMotor™ I/O using AniLink chip sets.

DIN{address}{channel} returns an unsigned 8 bit value, ranging from **0** to **255**. If the specified card or connection is not present, the function will return a value of **255**.

EXAMPLE 1: (reading the first 8 inputs of an OPTO-1 on Address A)

```
x=DINA0   'Assign first 8 inputs to "x"
```

EXAMPLE 2: (reading the second 8 inputs of an OPTO-1 on Address A)

```
x=DINA1   'Assign second 8 inputs to "x"
```

EXAMPLE 3: (reading the third input bit of an OPTO-1 on Address A)

```
x=DINA0 & 4   'Assign second 8 inputs to "x"
```

DOUT{port}{channel}{expression}

Output Byte to I/O Device

Related Command:

DIN

APPLICATION:	Input control
DESCRIPTION:	Output byte to Anilink digital peripheral
EXECUTION:	Immediate byte write to IIC link
CONDITIONAL TO:	Peripheral output attached to motor
LIMITATIONS:	Port = A . . H and Channel = 0 . . 63
REPORT COMMAND:	N/A
READ/WRITE:	Write only
LANGUAGE ACCESS:	Assignment to output peripheral only
UNITS:	Number
RANGE OF VALUES:	0 to 255
TYPICAL VALUES:	0 to 255
DEFAULT VALUE:	255
RELATED COMMANDS:	**DIN**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

NOTE: 8 bit data = Logical AND of expression with 255

The **DOUT{Address}{channel}, expression** command allows eight bits of data to be written to a peripheral I/O device such as the DIO-100 or OPTO-1 digital I/O expansion module. The value is transmitted via the AniLink communications channel. The "address" parameter must correspond with hardware address jumpers on the peripheral expansion card. The Addresses are designated as A, B, C, D, E, F, G, or H. The "channel" number, which may be from 0 to 63, is device specific. Typically it is 0 thru 8. See the specific peripheral user manual for specific details.

DIN{address}{channel} returns an unsigned 8 bit value, ranging from **0** to **255**. If the specified card or connection is not present, the function will return a value of **255**.

EXAMPLE 1: (sending data to the first 8 outputs of an OPTO-1 on Address A)

```
DOUTA0,255          'Sets first 8 outputs to 1
DOUTA0,0            'Sets first 8 outputs to 0
```

EXAMPLE 2: (setting value to specific bit output of an OPTO-1 on Address A)

```
x=DINA0             'Fist read state of the outputs
DOUTA0,x|4          'Set 3rd bit to 1
DOUTA0,x&251        'Set 3rd bit to 0
```

Related Commands

G

MP

MV

APPLICATION: Position Error Handling

DESCRIPTION: Maximum Allowable Following Error

EXECUTION: Immediate. Enforced each **PID** sample

CONDITIONAL TO: Trajectory in progress

LIMITATIONS: Torque mode has no position error

REPORT COMMAND: **RE**

READ/WRITE: Read and Write

LANGUAGE ACCESS: Assignment, expressions and conditional testing

UNITS: Encoder counts

RANGE OF VALUES: **0** to **8388607 (23 Bit UNSIGNED Value)**

TYPICAL VALUES: **1000**

DEFAULT VALUE: **1000**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The **E** command is used to set the maximum allowable *Position Error* in encoder counts. *Position Error* is the difference between the desired position, at any instant in time, and the actual position. The SmartMotor™ uses the position error to generate a torque by means of the **PID** filter. The more the error or deflection, the more torque the motor applies in attempt to correct.

E is primarily used as a safety measure, a programmable allowable error beyond which the motor recognizes it is outside of the domain of control you wish to enforce. If **E=100** is command and a position error of greater than 100 encoder counts occurs, the motor will be turned off. When the motor is turned off, the **Bo** (Motor-Off Bit) is set to **1**, and the **Be (Position Error Bit)** will be set to **1**. All closed-loop modes are bound by this **E** value. Non-closed loop modes such as **Torque Mode**, ignore the value of **E**.

The amount of Position Error is always proportional to the difference between commanded torque and load torque. The higher the commanded speed, the higher the position error will be. High Accelerations can lead to short duration high spikes in position error. The value for E should always be high enough to allow for acceleration and declaration ramps. It may be necessary to increase tuning gains to keep position error within reasonable limits for good dynamic operation.

EXAMPLE:

```
E=1234              'set maximum allowable error to 1234
```

If the motor dynamically ever exceeds 1234, it fault on Position error immediately.

ECHO

Echo Incoming RS-232 Data

Related Commands:

ECHO1

ECHO_OFF

ECHO_OFF1

APPLICATION:	Serial communications control
DESCRIPTION:	Motor echoes received channel 0 serial
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Applies to Channel 0 (Primary Com Port)
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	Motor defaults to **ECHO_OFF** (non-echo)
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The **ECHO** command causes the SmartMotor™ to re-transmit (or echo out) all serial bytes on the transmit line that were received on the receive line of the primary comm port. This retransmission occurs when the SmartMotor reads these bytes from the buffer, regardless of whether these bytes are command or individual data bytes. **ECHO_OFF** terminates the echo facility. **ECHO** can be issued to control a single motor communicating with a host terminal or any another serial device, as well as control groups of motors sharing series loop (daisy chain) serial communication lines.

ECHO is required to pass serial bytes though a motor to the next motor in a multi-drop serial daisy chain setup such as when the Add-A-Motor cables are used. It is also often used in single motor applications for transmit verification.

ECHO_OFF

Turn RS-232 Echo Off

Related Commands:

ECHO

ECHO_ON

ECHO_OFF1

APPLICATION: Serial communications control

DESCRIPTION: Motor does NOT echo received channel 0 serial characters

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Motor Defaults to **ECHO_OFF** (non-echo) off

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

ECHO_OFF causes the SmartMotor™ channel **0**, or primary, comm port to stop echoing. This is the default power-up state of any SmartMotor. No incoming channel **0** characters are re-transmitted. The command can be issued to control a single motor communicating with a host terminal or any another serial device, as well as control groups of motors sharing series or parallel serial communication I/O lines.

In order to automatically detect and differentiate between multiple motors on a serial daisy chain cable, the ECHO state can be alternately turned on and off to insure addressing is done properly.

Note: It is not possible to maintain communications on a serial chain without issuing ECHO.

ECHO1

Echo Incoming RS-485 Data

Related Commands:

ECHO

ECHO_OFF

ECHO_OFF1

APPLICATION:	Serial communications control
DESCRIPTION:	Motor echoes received channel 1 serial
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	**ECHO1** is off
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The **ECHO1** command causes the SmartMotor™ to re-transmit (or echo out) all serial bytes on the transmit line that were received on the receive line of the secondary comm port. This retransmission occurs when the SmartMotor reads these bytes from the buffer, regardless of whether these bytes are command or individual data bytes. **ECHO_OFF1** terminates the echo facility.

It is important to note that the channel 1 serial port is half-duplex RS485. It cannot simultaneously send and receive. Thus, when used directly as RS-485, the **ECHO1** command is not recommended.

Related Commands:

ECHO

ECHO_OFF

ECHO_OFF1

APPLICATION:	Serial communications control
DESCRIPTION:	Motor does NOT echo received serial 1 characters
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	**ECHO** is off
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

ECHO_OFF1 causes the SmartMotor™ channel 1 serial port to stop echoing. No incoming channel 1 characters are retransmitted. The command can be issued to control a single motor communicating with a host terminal or any another serial device, as well as control groups of motors sharing series or parallel serial communication I/O lines.

ELSE

IF-Structure command flow element

Related Commands:

ELSEIF exp

ENDIF

IF exp

APPLICATION: Program execution control

DESCRIPTION: Component of **IF** expression … **ELSE** …. **ENDIF** control block

EXECUTION: Immediate if exercised

CONDITIONAL TO: Value of associated **IF** expression

LIMITATIONS: Must reside with **IF** expression … **ENDIF** program control block

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

An **IF** expression … **ENDIF** control block may optionally include an **ELSE** statement to control execution when none of the test conditions are true. Suppose that you want the SmartMotor™ to do one thing if the variable **g=43**, and another if it isn't.

EXAMPLE:

```
IF g==43
      PRINT("Gee … 43!",#13)
ELSE
      PRINT("No 43 for me.",#13)

ENDIF
```

The first line checks to see if **g** is equal to **43**. If so, the string "Gee … 43!" is sent out the primary serial port. The **ELSE** in line 3 tells the SmartMotor what to do otherwise.

An **IF** control block can only have, at most, one **ELSE**. If such an **ELSE** exists and the language interpreter evaluates the **IF** expression to be false (zero) and there are no **ELSEIF** statements, then program will branch immediately to the statement following the **ELSE**. If there are **ELSEIF** expression clauses within the control block, all the **ELSEIF** clauses must precede the **ELSE** clause. In these cases the **ELSE** clause is only executed in if both the **IF** expression is false (zero) and all the **ELSEIF** expressions are false (zero).

Related Commands:

ELSEIF exp

ENDIF

IF exp

ELSE is analogous to the **DEFAULT** case for a **SWITCH** control block.

ELSE is not a valid terminal command, it is only valid within a user program.

EXAMPLE:

```
a=1                'PRINT("FALSE") is always executed
IF a==2
      PRINT("TRUE")
ELSE
      PRINT("FALSE")

ENDIF
```

EXAMPLE:

```
IF a==1     'only if a is NOT 1, 2, or 3
            'will GOSUB5 be executed.
      GOSUB2
ELSEIF a==2
      GOSUB3
ELSEIF a==3
      GOSUB4
ELSE
      GOSUB5
ENDIF
```

ELSEIF

IF-structure command flow element

Related Commands:

ELSE

ENDIF

IF exp

APPLICATION: Program execution control

DESCRIPTION: Alternate Evaluation of **IF** ..**ENDIF** control block

EXECUTION: **I**mmediate if exercised

CONDITIONAL TO: Value of associated **ELSEIF** expression

LIMITATIONS: Must reside with **IF** expression … **ENDIF** program control block

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

An **IF** expression, **ENDIF** control block may optionally include any number of **ELSEIF** expressions to perform multiple evaluations in a specified order. Suppose that you want the SmartMotor™ to do one thing if the variable **g=43**, another if **g=43000** and another if **g=-2**.

EXAMPLE:

```
IF g==43
      PRINT("Gee … 43!",#13)
ELSEIF g==43000
      PRINT("43 grand for me."#13)
ELSEIF g==-2
      PRINT("2?"#13)

ENDIF
```

The first line checks to see if **g** is equal to **43**. If so, the string "Gee … 43!" is sent out the primary serial port and the **IF** control block terminates. If **g** is not **43**, the program goes on to test if **g** is 43000. If it is, "43 grand for me." is sent out the primary serial port and the **IF** control block terminates. Similarly, if **g** is not **43000**, the program goes on to test if **g** is **-2**. If it is, "-2?" is sent out the primary serial port and the **IF** control block terminates.

An **IF** control block can have multiple **ELSEIF** statements. If such an **ELSEIF** clause exists and the language interpreter evaluates the **IF** expression to be false (zero) the program will branch immediately to first **ELSEIF** expression.

Related Commands:

ELSE

ENDIF

IF exp

If the associated expression is true, then the following clause is executed until an **ELSEIF**, **ELSE** or **ENDIF** is encountered and then execution branches to the **ENDIF** of the present **IF** control block. If the first **ELSIF** clause is not executed, then program execution continues at the next **ELSEIF** expression and so on until all the **ELSEIF** expressions have been tested. In the case all **ELSEIF**s have false expressions and an **ELSE** clause exists that clause will be executed.

The **ELSEIF** statement is similar to the **CASE** number case for a **SWITCH** control block. Note the difference - **ELSEIF** handles expressions, **CASE** only handle a fixed number.

ELSEIF is not a valid terminal command, it is only valid within a user program.

EXAMPLE:

```
a=3
IF a==2             'expression will be found false
      PRINT("222"
ELSEIF a==3         'expression will be found true
      PRINT("333"  'so "3333" will be printed.

ENDIF
```

EXAMPLE:

```
IF a==1             'only if a is NOT 1, 2, or 3
                    'will GOSUB5  be executed.
      GOSUB2
ELSEIF a==2
      GOSUB3
ELSEIF a==3
      GOSUB4
ELSE
      GOSUB5
ENDIF
```

Related Commands:

CTR

ENC1

APPLICATION: Encoder control

DESCRIPTION: Use internal encoder as the primary encoder

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT STATE: **ENC0**

FIRMWARE VERSIONS: 4.11 and higher

DETAILED DESCRIPTION:

The SmartMotor™ can accept inputs from either the internal integrated encoder or an external source. **ENC0** will cause the SmartMotor to read its position from the internal encoder, while **ENC1** uses the secondary (external) encoder. When **ENC0** is active, the external encoder input will be tracked by the **CTR** variable and **@P** will track the internal encoder.

EXAMPLE:

```
ENC1     'Servo from external encoder
ENC0      'restore default encoder behavior

ENC1     'Servo from external encoder
ENC0     'restore default encoder behavior
```

Related Commands:

ENC0

WARNING:

If the ENC1 command is issued without an external encoder connected both electrically to the A and B inputs and physically to the shaft, and connected properly, the shaft will run away with full speed and torque.

APPLICATION: Encoder selection control

DESCRIPTION: Swap internal and external encoder functions.

Use external encoder as the primary encoder.

The internal encoder is now associated with **CTR** value.

EXECUTION: Immediate

CONDITIONAL TO: External encoder attached to motor

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT STATE: **ENC0**

FIRMWARE VERSIONS: 4.11 and higher

DETAILED DESCRIPTION:

The SmartMotor™ can accept inputs from either the internal integrated encoder or an external source. The **ENC1** command will cause the SmartMotor to servo from the secondary (external) encoder channel, instead of the internal encoder. The internal encoder will likewise then be readable by way of the **CTR** variable. **@P** will rack the external encoder. The default mode of operating from the internal encoder is restored with the **ENC0** command.

If the external encoder is not connected or connected wrong, the motor may run away. If this happens, use the RP command to check the position. If by rotating the shaft you can change the position, then the encoder is connected, but the A and B signals likely need to be swapped to reverse the direction described by the quadrature phasing of the A and B signals.

EXAMPLE:

```
ENC1  'Servo from external encoder
ENC0  'restore default encoder behavior
```

END

End Program Code Execution

Related Commands:

RCKS

Rv

RUN

RUN?

UP

UPLOAD

Z

APPLICATION: Program execution control

DESCRIPTION: Terminates the user program execution

EXECUTION: Immediate

CONDITIONAL TO: Valid whether issued by host or user program

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

END terminates execution of a user program if running. **END** may be issued via serial communications channels or from within the user program itself. Each program must have a minimum of at least one **END** statement. The windows interface SMI scanner will not compile a source file without at least one **END** present. **END** only terminates the user program and internally resets the program pointer to the beginning of the program; no other state, variable, mode, or trajectory is affected.

The **SMI** program provides a speed bar button to send **END**. This is especially useful when something prevents the user from fully typing **END** at the terminal screen.

EXAMPLE:

```
IF Be END ENDIF    'terminate user program
                   'upon position error
```

Note: All PLS firmware Motors automatically issue END upon receiving any of the following error conditions:

Be (Position Error)

Bl (Left Travel Lmit)

Br (Right Travel Limit)

Bh (Over Temperature/RMS Over Current)

Please consult PLS firmware documentation for more details and options around this.

Related Command:

IF exp

ELSE

ELSEIF exp

APPLICATION: Program execution control

DESCRIPTION: **IF** expression … **ENDIF** control block terminator

EXECUTION: N/A

CONDITIONAL TO: There must exist a corresponding **IF** expression

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

*Every **IF** structure must be terminated with an **ENDIF***

Each control block commencing with **IF** expression … must have a corresponding **ENDIF** block exit statement. The program statement following **ENDIF** is the common exit point branched to upon processing the **IF … ENDIF** control block regardless of the execution path thought the control block at run time. There can only be one **ENDIF** statement for each **IF** statement. The common exit point following **ENDIF** is branched to upon the following:

1. Processing a true **IF** expression clause and encountering **ELSEIF**, **ELSE**, or **ENDIF**.
2. Processing a true **ELSEIF** expression and encountering another **ELSEIF**, **ELSE**, or **ENDIF**.
3. Processing an ELSE expression and encountering **ENDIF**.
4. If all **IF** and **ELSIF** expressions are false and there no **ELSE** clause.

ENDIF is not a valid terminal command, it is only valid within a user program.

EXAMPLE:

```
IF a==1
      PRINT("ok",#13)
ENDIF
PRINT("EXIT",#13)
```

ENDS

End SWITCH Statement

Related Command:

CASE number

DEFAULT

SWITCH exp

APPLICATION: Program execution control

DESCRIPTION: **SWITCH** expression … **ENDS** control block terminator

EXECUTION: N/A

CONDITIONAL TO: a corresponding **SWITCH** expression

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Each **SWITCH** expression must have a corresponding **ENDS** block exit statement. Any program statement immediately following **ENDS** is the common exit point branched to upon processing the **SWITCH . . . ENDS** control block regardless of execution path through the control block at run time. There can only be one **ENDS** statement for each **SWITCH** statement.

The common exit point following **ENDS** is branched to upon the following:

1. Upon encountering a **BREAK**
2. Upon encountering **ENDS**
3. The **SWITCH** expression value is not equal to any **CASE** number value and there is no **DEFAULT** statement label for the control block.

ENDS is not a valid terminal command, it is only valid within a user program.

EXAMPLE :

```
SWITCH x
      CASE 1 PRINT("x=1",#13) BREAK
      CASE 2 PRINT("x=2",#13) BREAK
      CASE 3 PRINT("x=3",#13) BREAK
ENDS
'This is the exit point for SWITCH...ENDS code block
```

EPTR=expression

Set Data EEPROM Pointer

Related Command:

VST

VLD

APPLICATION: EEPROM Data storage control

DESCRIPTION: Set user data EEPROM pointer

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: None

READ/WRITE: Write only. **EPTR** auto incriminated as used

LANGUAGE ACCESS: Assignment only

UNITS: EEPROM Address pointer

RANGE OF VALUES: **0** to **7999 <= v4.13, 0-32000 >= v4.15**

TYPICAL VALUES: **0** to **32000**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

EPTR sets the address location (pointer) within the Nonvolatile used data EEPROM for the data retrieval read **VLD**(variable, number) function and data storage write **VST**(variable, number) function. The **EPTR** value is write only, once it is set, **EPTR** auto-increments by 1, 2, or 4 with each read or write access to the physical EEPROM device according to the present data type.

EXAMPLE:

```
EPTR=4000   'set EPTR = 4000
VST(hh,1)   'store a 32 bit value
                 'EPTR is now 4004
VST(ab[7])  'store an 8 bit value
                 'EPTR is now 4005
VST(aw[7])  'store a 16 bit value
                 'EPTR is now 4007
VST(x,3)    'Store 3 consecutive variables, x,y,z
                 'EPTR is now 4007+(3*4) or 4019
VST(x,4)    'INVALID !!! EPTR remains 4019 !!!
```

Note: You cannot store consecutive variables past their group range. In other words, you can store any consecutive variables a-z or aa-zz or aaa-zzz within their groups only.

```
VST(aa,26)      'Perfectly Valid !!!
VST(aa,27)      'INVALID !!!
```

Related Command:

ES1000

APPLICATION:	EEPROM Read write Control
DESCRIPTION:	Set EEPROM read write rate to 400kz
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	None
READ/WRITE:	None
LANGUAGE ACCESS:	N/A
UNITS:	Bits per sec
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	1000
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

ES400 controls the transmit and receive bit rate while communicating between the EEPROMS and the microprocessor. There are two settings **ES400** and **ES1000**. **ES1000** is the preferable higher data transfer rate for read and writing user programs and data, and is the default data rate of version 4 and later SmartMotors™ and later. The **ES400** command is used with older EEPROMs. If you have an "older" EEPROMs and SmartMotors of differing versions, you may wish to consider upgrading the EEPROMS.

Note: The following applies to units prior to year 2000.

If you get an "F,"or failure, response to the **RCKS** command (report program check-sum) following a program download, you may wish to issue an **ES400** command from the terminal and try again. If **RCKS** now passes, you may have a slow EEPROM. In some cases you may need to make **ES400** the first program statement within a program, but as the command controls the speed at which the memory is read, the command really has little value in a program, and you may wish to consider upgrading the EEPROM.

Related Command:

ES1000

EXAMPLE:

The following simple test program may well abort if **ES400** is unreliable.

```
PRINT("TEST ES400 & ES1000")
a=1000
WHILE a
a=a-1
ES400                  'slower data rate
PRINT(#13,"ES400 ",a)
GOSUB5
ES1000                 'faster data rate
PRINT(#13,"ES1000 ",a)
GOSUB5
LOOP
PRINT(#13,"TEST RAN TO COMPLETION")
PRINT(#13,"NO DATA ERROR DETECTED")
END
C5
WAIT=100
c=a
b=a
IF c!=b
      PRINT("DATA PROBLEM - ABORT TEST")
ENDIF
RETURN                 'add many GOTO10 statements here
GOTO10                 'to fill up your program EEPROM
C10
PRINT(#13,"PROGRAM POINTER ERROR - ABORT TEST")
END
```

ES1000

Set EPROM Read/Write Speed

Related Command:

ES400

APPLICATION: EEPROM Read write Control

DESCRIPTION: Set EEPROM read write rate to 1000kz

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: EEPROM Read Write Capability

REPORT COMMAND: None

READ/WRITE: None

LANGUAGE ACCESS: N/A

UNITS: Bits per sec

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: 1000

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

ES1000 controls the transmit and receive bit rate while communicating between the EEPROMS and the microprocessor. There are two settings - **ES400** and **ES1000**. **ES1000** is the preferable higher data transfer rate for read and writing user programs and data, and is the default data rate of version 4 SmartMotors™ and later. The **ES400** command is used with older EEPROMs. If you have an "older" EEPROMs and Smart-Motors of differing versions, you may wish to consider upgrading the EEPROMs.

Note: the following applies to units prior to year 2000:

If you get an "F,"or failure, response to the **RCKS** command (report program check-sum) following a program download, you may wish to issue an **ES400** command from the terminal and try again. If **RCKS** now passes, you may have a slow EEPROM. In some cases you may need to make **ES400** the first program statement within a program, but as the command controls the speed at which the memory is read, the command really has little value in a program, and you may wish to consider upgrading the EEPROM.

Related Command:

ES400

EXAMPLE:

The following simple test program may well abort if **ES1000** is unreliable.

```
PRINT("TEST ES400 & ES1000")
a=1000
WHILE a
a=a-1
ES400                   'slower data rate
PRINT(#13,"ES400 ",a)
GOSUB5
ES1000                  'faster data rate
PRINT(#13,"ES1000 ",a)
GOSUB5
LOOP
PRINT(#13,"TEST RAN TO COMPLETION")
PRINT(#13,"NO DATA ERROR DETECTED")
END
C5
WAIT=100
c=a
b=a
IF c!=b
    PRINT("DATA PROBLEM - ABORT TEST")
ENDIF
RETURN                  'add many GOTO10 statements here
GOTO10                  'to fill up your program EEPROM
C10
PRINT(#13,"PROGRAM POINTER ERROR - ABORT TEST")
END
```

Related Command:

HA
KD
KG
KI
KL
KP
KS
KV

APPLICATION: Amplifier control

DESCRIPTION: Load buffered **PID** filter values into **PID** filter

EXECUTION: Next **PID** sample

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The servo tuning parameters, **KA**, **KD**, **KG**, **KI**, **KL**, **KP**, **KS**, and **KV**, are all buffered parameters. These parameters, once requested, take effect only when the **F** command is issued. This allows several parameters to be change at one time, without intermediate tuning states causing disruptions. Tuning parameters can be changed during a move profile, although caution is urged.

A default set of tuning parameters is in effect at power up or reset, but are optimized for an unloaded shaft. Different motor sizes have different optimal **PID** default gain values.

EXAMPLE:

```
KP=100              'initialize KP to a some value
F                   'load into present PID filter
G                   'start motion
WAIT=40000
KP=KP+10            'increment the present KP gain value`:
F                   'change into filter END
```

F=expression

Motor Function Control

Related Command:

None

APPLICATION:	Motor Function control
DESCRIPTION:	Miscellaneous commands
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	**RF**
READ/WRITE:	Write only
LANGUAGE ACCESS:	Assignment only
UNITS:	Number
RANGE OF VALUES:	0 to 15
TYPICAL VALUES:	0
DEFAULT VALUE:	0
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

F=value sets various functions or operational conditions of the motor. The value is Bit-weighted meaning that each binary bit is a on or off state for that particular function. As a result, it is also bit additive meaning that to turn on or off any selected function the appropriate bits must be set to 1 or 0. **F** is not assignable or readable. If you wish to rack it's value a shadow variable may be used.

Example:

```
x=2
F=x
x=2& 8
F=x
```

This allows you to keep track of the functions that are enabled or disabled via the **F** command.

The following page covers a description of each function.

Motor Function Control

Related Command:

None

F=1	Decelerate to stop on limit switch input (as opposed to just turning off)
F=2 *	Invert Commutation (Changes Shaft rotation)
F=4	Any Report commands transmit to Com 1 only. (Use with Extreme Caution)
F=8	Clear PID integral term at trajectory-end to avoid possible slow settling
F=16 *	Mode Cam positions are relative for each re-entry into CAM table (from either direction)
F=32 *	**GOSUB1** is issued under motor fault condition **C1** can not be called again prior to receiving a **RETURNF**
F=64 *	**GOSUB2** is issued on user input G transition from high to low **C2** can not be called again prior to receiving a **RETURNI**
F=128 *	Internal Slave Counter = base + dwell modulo while in CAM Mode
F=256 *	Set T.O.B. to be active for entire move profile.
F=512 *	Suppress T.O.B. until Slew Velocity has been reached
F=1024 *	Enables Port G to Index trigger latch function (only in SM2316D/DT >=4.93 firmware)

*** Note: Only Applies to >=v4.77 only……..**

Warning: C1 has priority over C2. C1 can be activated when in C2.

The F value can be changed on the fly while in an Interrupt subroutine to change its effect. An example would be turning off the G interrupt once in C2 to prevent any subsequent calls.

F Command is Binary Bit flag additive:

Example: F=21 would break down to **F=(16+4+1).** Motor would run CAM Mode relative, redirect print statements to port 1, and decelerate on limits.

Related Command:

None

Example using F=32 for Interrupt driven Fault routine

```
F=32        'Enable C1 Fault routine

MV          'Set to Velocity Mode
V=10000     'Set Speed
A=100       'Set Acceleration
G           'Start moving in Velocity Mode

END


C1     ' Fault Routine (Gets called on any of the following
faults)
      IF Be              ' Checking for error status bits
            PRINT(" Position Error",#13)
      ENDIF
      IF Bh
            PRINT(" Over Temp Error",#13)
      ENDIF
      IF Bi
            PRINT(" Over Current Error",#13)
      ENDIF
      IF Bl
            PRINT(" Left/Positive Travel Limit Error",#13)
      ENDIF
      IF Br
            PRINT(" Right/Negative Travel Limit Error",#13)
      ENDIF
      WHILE 1            'Wait for Motor Reset
            IF r==1      'If host sends r=1 via serial port
                  ZS     'Reset the motor
            ENDIF
            IF UAI==0    'If Input A gets rounded
                  ZS     'Reset the motor
            ENDIF
      LOOP
RETURNF                  'Return form Fault routine
```

Example using F=64 for Port G, C2 interrupt subroutine call

```
F=64  'Enable Port G interrupt routine
END
C2          ' Port G interrupt Routine
      PRINT(" Port G was grounded",#13)
RETURNI     ' Return  from Input Trigger
```

Example using F=64 for C2 subroutine call and F=1024 Index Re-direct for position capture

```
F=64+1024   'Enable Port G interrupt routine and Index Capture
Re-direct
END
C2    ' Port G interrupt Routine
      PRINT(" Port G was grounded",#13)
      PRINT(" Position captured at:",I,#13)
RETURNI 'Return  from Input Trigger
```

Related Command:

A
D
E
MC
MD
MFR
MP
MV
P
UG
UGI
UGO
V

APPLICATION: Trajectory control, Parameter Update

DESCRIPTION: Initiate or change trajectory parameters.

EXECUTION: Next **PID** sample

CONDITIONAL TO: Clearing of prior errors (in PLS firmware only)

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The **G** command stands for "Go" and is used to start motion or update buffered values such as Speed or acceleration.

A "**G"** command is required in each of the following cases:

1. Initiate an Absolute Move in **Mode Position** (**MP**)

```
V=10000 A=100 P=1234 MP G
```

2. Initiate a Relative Move in **Mode Position** (**MP**)

```
V=10000 A=100 D=4000 MP G
```

3. Initiate a Velocity in **Mode Velocity** (**MV**)

```
V=10000 A=100 MV G
```

4. Change to a new Velocity in **Mode Position** (**MP**) or **Mode Velocity (MV)**

```
V=10000 A=100 MV G  WAIT=1000 V=V*2 G
```

5. Change to a new Acceleration in **Mode Position** (**MP**) or **Mode Velocity (MV)**

```
V=10000 A=100 MV G  WAIT=1000 A=A*2 G
```

6 Initiate/Change an Electronic Gear Ratio in **Mode Follow with Ratio** (**MFR**),

```
MF0 MFMUL=1 MFDIV=10 MFR G
```

7 Initiate/Change an Electronic Gear Ratio in **Mode Step with Ratio** (**MSR**),

```
MF0 MFMUL=1 MFDIV=10 MSR G
```

8. Initiate **Cam Mode** (**MC**) :

```
MF0 MC1 G
```

Related Command:

A

D

E

MC

MD

MFR

MP

MV

P

UG

UGI

UGO

V

9. Begin **Host Mode** (**MD**).motion prior to filling all buffered data slots.

(See Users Guide for Host Mode)

10. Initiate a phase Offset Move while in Electronic Gear Ratio in either **Mode-Follow or Mode-Step**

```
MF0 MFMUL=1 MFDIV=10 MFR G  WAIT=2000 D=2000 V=100 G
```

On Power-Up, the Motor defaults to the Off state with MP (Mode Position buffered in with no Velocity or Acceleration values. As a result, if G is issued the motor will immediately servo in place.
Mode Follow (**MS1**, **MF1**, **MF2** and **MF4**), **Mode Step (MS), Mode Torque** (**MT**), and **Amplifier Mode** (**MD50**) are immediately active, they do not wait for any **G** command.

If a **G** command is transmitted and no motion results, any of the following may be the cause:

- **E=0** or too small
- **A=0** or **1**
- **V=0** or so small motion is not visible to naked eye
- Target position equals present position
- **D=0**
- **Bh=1** the motor is hotter than max permitted temperature **TH**
- **AMPS=0** or too small
- **T=0** or too small
- Motor is in **Torque Mode**
- **LIMD** is in effect and the "wrong" limit input switch is active
- Issued **MF0** or **MS0** instead of **MFn** or **MS**
- External encoder signal not present or not changing (in follow modes)
- Motor is part of a daisy chain that hasn't been properly set up
- Serial communications are good but target motor is not addressed
- Serial communications at incorrect baud rate
- Serial communications cable not attached or poorly connected
- Motor has no drive power
- Motor has a prior fault that needs to be cleared first (PLS firmware)
- Motor has no connections to limit switch inputs on boot-up and therefor has travel limit fault (PLS firmware)

Related Command:

A

D

E

MC

MD

MFR

MP

MV

P

UG

UGI

UGO

V

EXAMPLE:

```
A=100        'Set buffered Acceleration
V=10000      'Set buffered Velocity
P=1000       'Set buffered Position
MP           'Set buffered Position Mode
G            'load buffered move, Start Motion
```

To servo in place:

```
P=@P         'Set buffered position equal to actual position
G            'Servo in place
```

The execution time for **G** command varies with the computational burden of the mode or on the fly move. In the some cases, the **G** command computation may take longer than expected, and may result in motion profiles of poor quality or erroneous movement. This can happen in very tight loops that don't allow the **G** command to fully process with each cycle, such as the following:

EXAMPLE:

```
C10                'Place a label
  P=CTR            'Set position equal to CTR
  G                'Issue GO command
GOTO10             'Loop back to label
```

This type of code practice is not recommended because it forces a re-calculation over and over again and will cause abrupt jerks or small glitches in the move profile.

Related Command:

GETCHR1

LEN

LEN1

OCHN

WARNING:

The OCHN command will cause the SmartMotor to ignore incomming commands and can lock you out. It is a good idea to use the RUN? command during development. If you get locked out, you can recover by sending two capitol E's during the first 1/2 second after power up. This will cause the motor to abort its program and give you a chance to download a better one. The terminal software has utilites to do this.

APPLICATION: Serial communications control

DESCRIPTION: Fetch next character in channel 0 serial input buffer

EXECUTION: Immediate

CONDITIONAL TO: Requires that a character is in the buffer

LIMITATIONS: Must check if **LEN**>0 before using

REPORT COMMAND: N/A

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

GETCHR reads and removes the next available character in the channel 0 serial receive buffer. It is absolutely necessary to check that **LEN**>0 before issuing the **GETCHR** command.

Normally, the SmartMotor™ interprets incoming RS-232 data as commands. Sometimes, it is useful to prevent that from happening and instead, write a custom command interpreter. This is accomplished by re-opening the input channel in *data mode* with the **OCHN** command.

EXAMPLE:

```
C20                       'Place a label
  IF LEN>0        'Check to see that LEN>0
    c=GETCHR              'Get character from buffer
    IF c==69              'Check to see if it is an E
     END          'End the program
   ENDIF
 ENDIF
GOTO20                    'Loop back to C20
```

GETCHR1

Get Character From RS-485

Related Command:

GETCHR

LEN

LEN1

OCHN1

APPLICATION: Serial communications control

DESCRIPTION: Fetch next character in channel 1 serial input buffer

EXECUTION: Immediate

CONDITIONAL TO: Requires that a character is in the buffer

LIMITATIONS: Must check if **LEN1**>0 before using

REPORT COMMAND: N/A

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional test

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

GETCHR1 reads and removes the next available character in the channel 1 serial receive buffer. It is absolutely necessary to check that **LEN1**>0 before issuing the **GETCHR1** command.

Sometimes, it is useful to be able to accept special commands and/or data over the RS-485 port such as might come from a light curtain or a bar code reader. This is accomplished by opening the input channel in data mode with the **OCHN1** command.

EXAMPLE:

```
C20                        'Place a label
  IF LEN1>0          'Check to see that LEN>0
    c=GETCHR1              'Get character from buffer
    IF c==69               'Check to see if it is an E
     END             'End the program
   ENDIF
 ENDIF
GOTO20                     'Loop back to C20
```

GOSUB{number}

Subroutine Call

Related Command:

C{number}

GOTO{number}

STACK

APPLICATION:	Program execution control
DESCRIPTION:	Perform subroutine beginning at C*number*
EXECUTION:	Immediate
CONDITIONAL TO:	C number previously defined
LIMITATIONS:	**GOSUB0** to **GOSUB999**
	nesting msut be <=6 levels deep !!!
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The **GOSUB{number}** command redirects program execution to a subroutine of the program marked with a label **C{number}**. The end of every subroutine is marked by the **RETURN** statement, which causes execution to return to the line following the corresponding **GOSUB{number}** command. Subroutines may call further subroutines; this is called nesting. There may be as many as a thousand **GOSUB**s but they may be nested only up to six deep. A subroutine may call itself, which is called recursion but is highly discouraged because it can lead to a stack overflow or nesting limit. A counter, conditional test or some other scheme can prevent exceeding the nesting limit.

The **STACK** control flow command explicitly and deliberately destroys the **RETURN** address history. Thus, if you issue **STACK**, take care that the program execution does not encounter a **RETURN** before the next **GOSUB**.

Subroutines present a great opportunity to partition and organize your code.

The **GOSUB** command is valid from both the serial channels and within the a user program. Do not, however, issue **GOSUB{number}** unless the corresponding **C{number}** label exists within the stored program. Otherwise you willg et a memory pointing error.

Note: If an attempt to issue a nonexistent GOSUB call is done via serial port, the motor will respond with "+/-" which basically means a memory error.

Related Command:

C{number}

GOTO{number}

STACK

EXAMPLE:

```
GOSUB20            'run subroutine 20
GOSUB21            'run subroutine 20
a=3
GOSUB25            'run subroutine 20
END                      'End code execution

C20                      'nested subroutine
  GOSUB30
  PRINT("20",#13)
RETURN
C21                      'nested subroutine
  GOSUB30
  PRINT("21",#13)
RETURN
C25                      'recursive subroutine
  PRINT(" 25:",a)
  a=a-1
  IF a==0
      RETURN
  ENDIF
  GOSUB25
RETURN
C30                      'normal subroutine
  PRINT(#13,"Subroutine Call ")
RETURN
```

The output will be as follows:

```
Subroutine Call 20

Subroutine Call 21

 25:3 25:2 25:1
```

In the above program example you can issue GOSUB20, GOSUB21, GOSUB25 or GOSUB30 from the terminal as well.

GOTO{number}

Branch Program Flow to a Label

Related Command:

BREAK

C{number}

ELSE

DEFAULT

GOSUB{number}

NOTE:

Extensive use of IF statements and GOTOs can quickly make your programs impossible to read or debug.

Learn to organize your code with one main loop using a GOTO and write the rest of the program with subroutines.

APPLICATION:	Program execution control
DESCRIPTION:	Branch program execution to statement **C**{number}
EXECUTION:	Immediate
CONDITIONAL TO:	**C**{number} previously defined
LIMITATIONS:	**GOTO0** to **GOTO999**
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The **GOTO{Number}** command unconditionally redirects program execution control to another part of the program marked by the label **C{Number}**.

The **GOTO{Number}** command is valid from both the serial channels and within a user program. Take care, however, not to issue a **GOTO{Number}** command unless the corresponding **C{Number}** label exists witihn the stored program.

Novice programmers use **IF** statements and **GOTO**s to create elaborate and sophisticated programs that quickly become impossible to read or debug. Force yourself to use **GOSUB**s for program control. You'll be glad you did.

EXAMPLE: (download the following program)

```
C0                  'Place main label
  IF UAI==0
    PRINT("Input A Low",#13)
  ENDIF
GOTO0               'GOTO allows program to run forever
END
```

I (capital i)

Encoder Index Pulse Location

Related Commands:

Bi

Bx

RBi

RBx

APPLICATION:	Hardware Index Capture
DESCRIPTION:	Encoder value latched by hardware index capture
EXECUTION:	Immediate
CONDITIONAL TO:	Index previously captured
LIMITATIONS:	High velocity at time of capture will create a systematic offset error
REPORT COMMAND:	**RI**
READ/WRITE:	Read only
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Encoder counts
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-2147483648** to **2147483647**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

I (capital i) is the function that stores the last hardware latched encoder index position. It can be read from a host with the **RI** command, or it can be read by the program with a line such as **a=I**. Only after it is read by either of these means, will the SmartMotor™ be looking for the next Index event. The host or the program can monitor for the event by reading the flag, **Bi**. **Bi** will read as zero until an index is latched, at which time **Bi** will be set to one. **Bi** is set to zero when the index position is read or accessed.

The commands **RI** and **PRINT(I,#13)** report the captured index value through the primary serial channel. **PRINT1(I,#13)** reports through the channel 1 serial port. All three commands reset the **Bi** flag to zero. Assignments such as *variable*=**I** also assign the captured value and reset the **Bi** flag to zero. If **Bi** is zero at the time the **I** value is accessed, the previously captured index value is returned again.

The index is a physical reference mark on the encoder. It is also referred to as a **Z** pulse, marker pulse, and sometimes combinations of all three names. Its most widely used in homing sequences requiring a high degree of repeatability.

Related Commands:

Bi

Bx

RBi

RBx

EXAMPLE: (homing against a hard stop with Index reference)

```
AMPS=100            'Current limit 10%
O=0                 'Declare this home
MP                  'Set Mode Position
A=100               'Set Acceleration
V=100000            'Set Velocity
P=-1000000          'Move negative
G                   'Start Motion
WHILE Bt            'Wait for motion fault
  IF Bi             'If Index Pulse Seen
    a=I             'Record Index Position
  ENDIF
LOOP                'Loop back to wait
O=-a                'Last Index is Home
P=0                 'Move to New Home
G                   'Start Motion
AMPS=1023           'Restore power
```

Note: >=v4.95 has the ability to redirect Port G to the Index register input trigger allowing high speed position capture via Port G this capture time occurs at CMOS level and is typically around 3 to 5 microseconds.

All the same rules apply to arming and clearing the index as stated above.

The Re-Direct to Port G is accomplished with the F command. See F= in this programmers guide for more detail.

Related Commands:

ELSE

ELSEIF

ENDIF

APPLICATION:	Program execution control
DESCRIPTION:	Conditional run time program execution
EXECUTION:	Test expression and take action as coded
CONDITIONAL TO:	Program execution branch if expression is zero or false
LIMITATIONS:	Requires corresponding **ENDIF** Can be executed only from within user program
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The **IF** statement is the basic means by which an executing program can make a choice between alternative execution paths at runtime. In its simplest form the **IF** control block consists of:

IF *(expression) evaluates as non-zero*
Run the code below the "IF" command
ENDIF

Every "IF" structure must be terminated with an "ENDIF".

Expression is a test condition, Both mathematical comparisons and boolean logic bit wise comparisons can be used. Each must evaluate to be true.

IF a==b	(If a equals b)	IF a!=b	(If a does not equal b)
IF a<b	(If a is less than b)	IF a<=b	(If a is less than or equal to b)
IF a>b	(If a is greater than b)	IF a>=b	(If a is greater than or equal to b)
IF a&b	(If a AND b, bit-wise)	IF a|b	(If a OR b, bit wise comparison)
IF a	(If a does not equal zero, common shortcut to IF a==1)		

All above examples must be True to allow code beginning below the IF command to run. If they are not true, the code execution will jump down to the nearest ELSE, ELSEIF or ENDIF and continue from there.

Related Commands:

ELSE

ELSEIF

ENDIF

Example 1: Simple case of: IF true, run some code.

```
IF @P>12345 'If Position is above 12345

      PRINT("position is greater than 12345",#13)
ENDIF
            'This is the next line of code to be executed
            'whether it is true or not.
```

Example 2: If true, run some code, ELSE if false run some other code...

```
IF @P>12345       'If Position is above 12345
      PRINT("position is greater than 12345",#13)
ELSE              'If it is no true
      PRINT("position is not greater than 12345",#13)
ENDIF
'This is the next line of code to be executed
```

Example 3: If true, run some code, else if something else is true......

```
IF @P>12345       'If Position is above 12345
      PRINT("position is greater than 12345",#13)
ELSEIF @P==0      'If Position equals zero
      PRINT("position is at zero",#13)
ENDIF
'This is the next line of code to be executed
'even if position is not at zero and
'not greater than 12345.
```

Example 4: Test for two conditions and default to another line of code:

```
IF @P>100         'If Position is above 100
      PRINT("position is greater than 100",#13)
ELSEIF @P<=0      'If it less than or equal to zero
      PRINT("position is <= to zero",#13)
ELSE
      PRINT("position is between zero and 100",#13)
ENDIF
```

(Continued on next page)

Related Commands:

ELSE

ELSEIF

ENDIF

Every "IF" structure must be terminated with an "ENDIF".

Example 5: Binary Bit Mask Comparison:

```
a=10  'binary 1010
b=5   'binary 0101
c=7   'binary 0111
d=1   'binary 0001
e=0   'binary 0000

IF a&2      'Compare "a" and 2 as binary numbers bit for it.
      PRINT("This is true because 2 is 0010",#13)
ENDIF
IF a&d      'Are any bits in common with a AND d?
      PRINT("This will never PRINT",#13)
ENDIF
IF a|b      'Are there any bits that are 1 in either number?
      PRINT("This will print",#13)
ENDIF
IF d|e      'even though e is zero, d is non-zero:
      PRINT("This will print",#13)
ENDIF
IF b&c
      PRINT("This is true",#13)
ENDIF
END
```

Related Commands:

F

RKA

KD

KG

KI

KL

KP

KS

KV

APPLICATION:	**PID** filter control
DESCRIPTION:	Acceleration feed forward gain
EXECUTION:	Buffered pending an **F** command
CONDITIONAL TO:	N/A
LIMITATIONS:	Must be positive
REPORT COMMAND:	**RKA**
READ/WRITE:	Read write
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	N/A
RANGE OF VALUES:	**0** to **65535**
TYPICAL VALUES:	**0 to 3000**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

KA sets the buffered acceleration feed forward gain. The acceleration feed forward term helps the **PID** filter to cope with the predictable effects of acceleration and inertia.

The **KA** gain factor is only applied in position (**MP**) and velocity (**MV**) moves. Issuing a new **KA** parameter is not effective until it is loaded into the present **PID** filter by the **F** command. The default value for **KA** is **0**, and acceptable values range from **0** to **65,535**.

It is difficult or impossible to tune **KA** in low inertia systems. Even in high inertia systems it can be a challenge to observe the benefit during very short acceleration periods. It is best to rely on the host tuning utility for assistance if it is thought that **KA** could be useful.

PRINT(KA,#13) and RKA both report the value of **KA** through the primary serial port, while **PRINT1(KA, #13)** sends it out channel 1. **KA** is valid with any expression, and can be treated as if it were any read-write variable. The motion or servo characteristics are unaffected until **KA** is applied by the **F** function.

EXAMPLE:

```
KA=200        'set buffered acceleration feed forward
F             'update PID filter
```

Related Commands:

KA

KG

KI

KL

KP

KS

KV

APPLICATION:	**PID** filter control
DESCRIPTION:	Derivative gain
EXECUTION:	Buffered pending an **F** command
CONDITIONAL TO:	N/A
LIMITATIONS:	Must be positive
REPORT COMMAND:	**RKD**
READ/WRITE:	Read write
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	N/A
RANGE OF VALUES:	**0** to **65535**
TYPICAL VALUES:	**400** to **2000**
DEFAULT VALUE:	Motor size dependent
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

KD sets the value of the derivative gain of the **PID** filter. If the **PID** filter gives stable performance, **KD** is usually the vibration absorbing, or damping, term.

For any stable **KP** value there is an optimum **KD** value, prior to and beyond which the motor will be unstable. An effective way to tune the filter, therefore, is to repetitively raise the **KP** value and then run the **KD** term up and down to find the local optimum. The point at which the **KD** term cannot stabilize the servo is the point where **KP** has gone too far. To test each setting twist the shaft of the motor and let it go while looking for abrupt and resolute response. The host level tuning utility can be useful in finding the optimum. The **F** command must be issued for a new buffered **KD** parameter to take effect. Typically a KD of ~10x KP is a good starting point for any given KP<300.

PRINT(KD,#13) and **RKD** both report the value of **KD** through the primary serial port, while **PRINT1(KD, #13)** sends it out channel 1. **KD** is valid with any expression, and can be treated as if it were any read-write variable. The motion and servo characteristics are unaffected until **KD** is applied by the **F** function.

EXAMPLE:

```
KD=2000            'set buffered derivative gain
F                  'update PID filter
```

KG=expression

PID Gravity Compensation

Related Commands:

KA

KD

KI

KL

KP

KS

KV

APPLICATION:	**PID** filter control
DESCRIPTION:	Gravitational gain
EXECUTION:	Buffered pending an **F** command
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	**RKG**
READ/WRITE:	Read write
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	N/A
RANGE OF VALUES:	**-8388608** to **8388607**
TYPICAL VALUES:	**0**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	**ALL**

DETAILED DESCRIPTION:

KG sets the gravity compensation term of the **PID** filter.

Simple **PID** filters are ill equipped where a constant force is asserted on the system. An example of such a constant force is that induced by gravity acting on a vertically moving axis. The **KG** term exists to offset the **PID** filter output in a way that removes the effect of such constant forces.

The best way to set **KG** is to turn **KP** and **KI** to zero and servo in place. The load will want to fall, but hold it in place. Issue increasingly positive or increasingly negative **KG** parameters until the load barely holds. Record that value and continue increasing the parameter until the load starts to go up. Now record this value. The optimum **KG** value is the average of these two.

Valid values for **KG** are integers from **-8388608** to **8388607.** As a result, you may not see much of an effect until **KG** is greater than one million in magnitude. However, extremely higher magnitudes values risks rapid pulse width modulation (PWM) saturation (uncontrollable servo behavior). The default value is **0**.

PRINT(KG,#13) and **RKG** both report the value of **KG** through the primary serial port, while **PRINT1(KG, #13)** sends it out channel 1. **KG** is valid with any expression, and can be treated as if it were any read-write variable. The motion and servo characteristics are unaffected until **KG** is applied by the **F** function.

EXAMPLE :

```
KG=10000000        'Set buffered Gravity Term
F             'Update Filter
```

Related Commands:

KL

KA

KD

KG

KP

KS

KV

APPLICATION: **PID** filter control

DESCRIPTION: Integral gain

EXECUTION: Buffered pending an **F** command

CONDITIONAL TO: N/A

LIMITATIONS: Must be positive, total integral limited by **KL**

REPORT COMMAND: **RKI**

READ/WRITE: Read write

LANGUAGE ACCESS: Assignment, expressions and conditional testing

UNITS: N/A

RANGE OF VALUES: **0** to **32767**

TYPICAL VALUES: **0** to equal that of present KP

DEFAULT VALUE: Motor size dependent

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The **KI** term sets the integral gain of the **PID** filter. The integral compensator is not for stability. Raising it too far will cause the motor to become unstable. The **KI** command is designed to compensate for friction in the system. Since the amount of power sent to the motor is proportional to the distance it is from its target position, there comes a time, close to the target, where the small position error is creating too small of a torque for the motor to reach the final target.

The integral term of the **PID** filter is generated by taking the sum of the position error of every sample and then multiplying by **KI**. As such, it creates a force that is a function of error and time. As time passes (a few milliseconds) and the control sees that a correction is not being made, it boosts the signal. This boost occurs at a rate set by the **KI** parameter. While you are tuning your motor for stability, it is probably a good idea to set **KI** to zero, and then later bring it up until you see that it reliably compensates for the friction of your system. The **F** command must be issued for a new buffered **KI** parameter to take effect and **KL**, the protective upper limit, must be high enough to allow **KI** to do its job.

PRINT(KI,#13) and **RKI** both report the value of **KI** through the primary serial port, while **PRINT1(KI, #13)** sends it out channel 1. **KI** is valid with any expression, and can be treated as if it were any read-write variable. The motion and servo characteristics are unaffected until **KI** is applied by the **F** function.

EXAMPLE:

```
KI=250                  'Set buffered integral gain
F                       'Update Filter
```

KL=expression
PID Integral Limit

Related Command:

KA

KD

KG

KI

KP

KS

KV

APPLICATION:	**PID** filter control
DESCRIPTION:	Integral limit
EXECUTION:	Buffered pending an F command
CONDITIONAL TO:	N/A
LIMITATIONS:	Must be positive
REPORT COMMAND:	**RKL**
READ/WRITE:	Read write
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	N/A
RANGE OF VALUES:	**0** to **32767**
TYPICAL VALUES:	**5** to **200**
DEFAULT VALUE:	Motor size dependent
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The **KL** term sets a limit on the effects of the **KI** term. Since the **KI** integrates the position error over time, it can ultimately dominate the **PID** equation. **KL** sets an upper limit on what the **KI** term can be.

Physically speaking, the **KI** term will raise the power to the servo as a function of time. If there is something other than friction blocking the servo and it is unable to move, the amount of torque given to the motor over time can quickly become unreasonably large. It is therefor a good idea to keep **KL** as low as possible while still allowing the **KI** term to effectively contend with friction. The **F** command must be issued for a new buffered **KL** parameter to take effect.

PRINT(KL,#13) and **RKL** both report the value of **KL** through the primary serial port, while **PRINT1(KL, #13)** sends it out channel 1. **KL** is valid with any expression, and can be treated as if it were any read-write variable. The motion and servo characteristics are unaffected until **KL** is applied by the **F** function.

EXAMPLE:

```
KL=1500                    'Set buffered integral limit
F                          'Update Filter
```

KP=expression

PID Proportional Compensation

Related Command:

KA

KD

KG

KI

KL

KS

KV

APPLICATION:	**PID** filter control
DESCRIPTION:	Proportional gain
EXECUTION:	Buffered pending an **F** command
CONDITIONAL TO:	N/A
LIMITATIONS:	Must be positive
REPORT COMMAND:	**RKP**
READ/WRITE:	Read write
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	N/A
RANGE OF VALUES:	**0** to **32767**
TYPICAL VALUES:	**40** to **300**
DEFAULT VALUE:	Motor size dependant
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The **KP** command is used to set the gain of the proportional parameter of the **PID** filter. Any new value of **KP** is held in a buffer until an **F** command is issued.

The higher the **KP** the stiffer the motor will be. At some point the added stiffness will cause the motor to become unstable. If moving the **KD** value up and down cannot stabilize the servo, then the **KP** value is too high and must be reduced.

PRINT(KP,#13) and **RKP** both report the value of **KP** through the primary serial port, while **PRINT1(KP, #13)** sends it out channel 1. **KP** is valid with any expression, and can be treated as if it were any read-write variable. The motion and servo characteristics are unaffected until **KP** is applied by the **F** function.

EXAMPLE:

```
KP=250                  'Set buffered proportional gain
F                       'Update Filter
```

Related Command:

KA

KD

KG

KI

KL

KP

KV

APPLICATION:	**PID** filter control
DESCRIPTION:	Inertial load gain
EXECUTION:	Buffered pending an **F** command
CONDITIONAL TO:	N/A
LIMITATIONS:	Must be positive
REPORT COMMAND:	**RKS**
READ/WRITE:	Read write
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	N/A
RANGE OF VALUES:	**0** to **255**
TYPICAL VALUES:	**1**
DEFAULT VALUE:	**1**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The **KS** term of the extended **PID** filter will sometimes allow the SmartMotor™ to handle inertial ratios in excess of the traditional 5:1 or 10:1. This reflected load to rotor inertia ratio is often sighted as a traditional limit for dependable servo motor application. The **KS** term represents the number of sample periods used to form the integration of the **KD** term. By raising the **KS** value beyond one, a latency is developed within the response vector of the **PID** equation's differential element. Since this reduces the rate at which the current error switches sign, it allows the motor to apply its available torque more decisively. This is also useful in situations where the mechanical time constant of the motor/load system is longer than the **PID** period by several orders of magnitude. Such systems can be very difficult to stabilize with a traditional **PID** filter.

If your application has an inertial ratio of greater than 5:1, experiment with raising **KS** above 1. Your ear will provide a good method of judgment; listen for a range **KS** values which provide relaxed but decisive motor response across the velocity and acceleration regions required by your application.

PRINT(KS,#13) and **RKS** both report the value of **KS** through the primary serial port, while **PRINT1(KS, #13)** sends it out channel 1. **KS** is valid with any expression, and can be treated as if it were any read-write variable. The motion and servo characteristics are unaffected until **KS** is applied by the **F** function.

EXAMPLE :

```
KS=5          'Set buffered differential sample rate
F             'Update Filter
```

Related Command:

KA

KD

KG

KI

KL

KP

KS

APPLICATION:	**PID** filter control
DESCRIPTION:	Velocity feed forward gain
EXECUTION:	Buffered pending an F command
CONDITIONAL TO:	N/A
LIMITATIONS:	Must be positive
REPORT COMMAND:	**RKV**
READ/WRITE:	Read write
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	N/A
RANGE OF VALUES:	0 to 32767
TYPICAL VALUES:	0 to 400
DEFAULT VALUE:	0
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

KV sets the gain for the velocity feed forward element of the extended **PID** filter. The velocity feed forward element can be thought of as a dynamically proportional adjustment to the **PID** filter required by the latency of the digital filter with respect to time. A zero value for **KV** disables the term within the filter.

If you put the SmartMotor™ into at a relatively high speed velocity move and monitor the position error with the Status Monitor, you will see a constant position error. Issue a series of successively larger **KV** parameters followed by **F** commands and watch the error reduce to zero.

The default value for **KV** is zero, acceptable values range from **0** to **65,535**. Typically useful values range from **0** to **2000**. Current values can be read back with **RKV**.

PRINT(KV,#13) and **RKV** both report the value of **KV** through the primary serial port, while **PRINT1(KV, #13)** sends it out channel 1. **KV** is valid with any expression, and can be treated as if it were any read-write variable. The motion and servo characteristics are unaffected until **KV** is applied by the **F** function.

EXAMPLE :

```
KV=1000             'Set buffered velocity feed forward
F                   'Update Filter
```

Related Command:

GETCHAR

GETCHAR1

LEN1

APPLICATION: Communication control

DESCRIPTION: Number of characters in serial host (channel 0) receive buffer

EXECUTION: Immediate

CONDITIONAL TO: Host communication channel open

LIMITATIONS: Maximum buffer length is 16 characters

REPORT COMMAND: None

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Number of available characters

RANGE OF VALUES: **0** to **16**

TYPICAL VALUES: **0** to **16**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

LEN returns the number of characters placed in the serial communications channel 0 receive buffer which are still awaiting to be processed. A serial channel in **COMMAND** mode will typically return **LEN** as 0, but a serial channel in **DATA** mode may well return a non zero value. Testing the value of **LEN** is a good way to see if there is any character for **GETCHR** to fetch.

EXAMPLE:

```
DAT                  'Set serial channel 0 to DATA mode
i=0
IF LEN               'any data received?
      GOSUB5         'if so process data
ENDIF
END
C5
ab[i]=GETCHR         'read and store in data
                     'process incoming data
i=i+1                'maintain reference index
RETURN
```

From the above example, "i" will be equal to LEN.

Related Command:

GETCHAR

GETCHAR1

LEN

APPLICATION:	Communication control
DESCRIPTION:	Number of characters in channel 1 serial receive buffer
EXECUTION:	Immediate
CONDITIONAL TO:	Host communication channel open
LIMITATIONS:	Maximum buffer length is 16 characters
REPORT COMMAND:	None
READ/WRITE:	Read only
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Number of available characters
RANGE OF VALUES:	**0** to **16**
TYPICAL VALUES:	**0** to **16**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

LEN1 returns the number of characters placed in the serial communications channel 1 receive buffer which are still awaiting to be processed. A serial channel in COMMAND mode will typically return LEN1 as 0, but a serial channel in DATA mode may well return a non zero value. Testing the value of LEN1 is a good way to see if there is any character for GETCHR to fetch.

EXAMPLE:

```
DAT1                    'make serial channel 1 DATA mode
i=0
IF LEN1                 'any data received ?
      GOSUB5            'if so process data
ENDIF
END
C5
ab[i]=GETCHR1           'read and store in data
                        'process incoming data
i=i+1                   'maintain reference index
RETURN
```

From the above example, "i" will be equal to LEN.

Related Command:

LIMH

LIML

LIMN

APPLICATION:	Travel Limit switch controL
DESCRIPTION:	Limit switches have directional property
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT PROPERTY:	Limits are non directional
FIRMWARE VERSIONS:	4.15 and 4.40 (non-PLS firmware only)

DETAILED DESCRIPTION:

LIMD (Limit Directional) specifies the way the SmartMotor™ responds to a **G** command while any limit input is active.

LIMD prevents motion further into or past the detected limit. **LIMD** can be cancelled by **LIMN** (Limit non-directional), which allows movement further into the limit. Neither of these commands change the response of the motor when it encounters a limit after already in motion.

Basic Effects of **LIMD** are as follows:

If the Positive Limit is active and the motor is commanded in the positive direction, it will fail to move.

If the negative limit is active and the motor is commanded in the negative direction, the motor will fail to move.

In both cases above, **LIMD** has prevented further motion beyond the detected travel limit.

In contrast, if the negative limit is active and motion is commanded in the positive direction, motion will be allowed.

If the positive limit is active and motion is commanded in the negative direction, motion will be allowed.

Note: LIMD behavior is applicable to all modes of operation.

LIMH

Travel Limits Active High

Related Command:

LIMD

LIML

LIMN

UCP

UDM

APPLICATION: Travel Limit Switch Control

DESCRIPTION: Limits are active high to stop motion

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT PROPERTY: Limits are active low

FIRMWARE VERSIONS: 4.15 and 4.40. (not available in PLS firmware)

DETAILED DESCRIPTION:

The limit switches are associated with the I/O C and I/O D pins. Following a power up or reset (the **Z** command), the limit inputs are active **LOW** by default. This means if the logic state goes low, the motor will stop.

LIMH defines the limit inputs to be active **HIGH**. This means if the logic state level goes high, the motor will stop.

NOTE: The limit input pins have 5K Ohm pull-ups meaning they are seen as logic high when there is no connection to them.

LIML defines them back to active low.

Associated with the limit switches are the system flags:

Hardware Travel Limit Overview				Status Bits		Command to Clear Historical Bit	Command to Disable Travel Limit Input	Command to Enable Travel Limit Input
Port	Pos/Neg	Plus/Minus	Left/Right	Real Time	Historical			
Port C	Positive	PLUS	RIGHT	**Br**	**Bp**	**Zr, or ZS**	**UCI or UCO**	**UCP**
Port D	Negative	MINUS	LEFT	**Bl**	**Bm**	**Zl, or Zs**	**UDI or UDO**	**UDM**

Note: PLS firmware defaults to LIMH with no option to change it.

Please consult PLS firmware documentation for more information.

Related Command:

LIMD

LIMH

LIMN

UCP

UDM

APPLICATION: Limit switch control

DESCRIPTION: Limit switches are active low

EXECUTION: Immediate

CONDITIONAL TO: N//A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT PROPERTY: Limit switches are active low

FIRMWARE VERSIONS: 4.15 and 4.40

DETAILED DESCRIPTION:

The limit switches are associated with the I/O C and I/O D pins. Following a power up or reset (the **Z** command), the limit inputs are active **LOW** by default. This means if the logic state goes low, the motor will stop.

LIML defines the limit inputs to be active **Low**. This means if the logic state level goes low, the motor will stop.

NOTE: The limit input pins have 5K Ohm pull-ups meaning they are seen as logic high when there is no connection to them.

LIMH defines them to active High.

Associated with the limit switches are the system flags:

Hardware Travel Limit Overview				Status Bits		Command to Clear Historical Bit	Command to Disable Travel Limit Input	Command to Enable Travel Limit Input
Port	Pos/Neg	Plus/Minus	Left/Right	Real Time	Historical			
Port C	Positive	PLUS	RIGHT	**Br**	**Bp**	**Zr, or ZS**	**UCI or UCO**	**UCP**
Port D	Negative	MINUS	LEFT	**Bl**	**Bm**	**Zl, or Zs**	**UDI or UDO**	**UDM**

Note: PLS firmware defaults to LIMH with no option to change it.

Please consult PLS firmware documentation for more information.

Related Command:

LIMD

LIML

LIMH

APPLICATION:	Limit switch control
DESCRIPTION:	Limit switches non directional
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT PROPERTY:	Limit switches are non directional
FIRMWARE VERSIONS:	4.15 and 4.40 (not available in PLS firmware)

DETAILED DESCRIPTION:

LIMN (Limit NON-Directional) specifies the way the SmartMotor™ responds to a **G** command while any limit input is active.

LIMN means that if you are on a limit switch (if it is active at the time). The motor will still be allowed to move in the same direction upon receiving another G (go) command.

Basic Effects of **LIMN** are as follows:

If the Positive Limit is active and the motor is commanded in the positive direction,
it will still be able to move

If the negative limit is active and the motor is commanded in the negative direction, it will still be able to move.

Note: LIMN behavior is applicable to all modes of operation.

Related Command:

LOCKP

LOCKPROM

RCKS

RUN

RUN?

UP

UPLOAD

APPLICATION: User program EEPROM control

DESCRIPTION: Receive and store SmartMotor™ executable program

EXECUTION: Immediate

CONDITIONAL TO: User program EEPROM present

LIMITATIONS: EPPROM capacity is limited to 8k, 16k, or 32k

REPORT COMMAND: **UP, UPLOAD, RCKS**

READ/WRITE: EEPROM is read write unless "locked"

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

This command is intended to be used in custom terminal software

LOAD is used by a terminal to download a compiled program file and store it within the USER PROGRAM EEPROM of the SmartMotor. The **LOAD** command causes a SmartMotor to load all incoming host communications into program memory up to the first occurrence of the ASCII character 255. Program sizes can be as great as 32k. This command is mainly used by host utilities, which also compiles the program before download.

LOAD does not terminate the present motion mode or trajectory, change motion parameters such as **E**, **A**, **V**, **KP** etc, or alter the present value of the user variables.

If the motor does not receive the ASCII 255 byte sometime after the **LOAD** command, the motor will continue to store incoming serial bytes directly to the Program EEPROM; During this time you are likely to be confused by the motor's apparent lack of response to your commands. The only way to terminate this condition is to transmit ASCII 255s or to reset the power.

Note: The SMI (SmartMotor Interface) software package is adjusted to take care of this automatically.

By using the "**LOAD**" command you can download from any controller/HMI/PLC or PC based program capable of storing an ASCI text file. For any given motor that is actively addressed, (i.e. you are talking to it and it responds) If you issue the **LOAD** command to the motor, it immediately goes into a memory-write mode while checking all incoming data. Every ASCII character that is received after the **LOAD** command is issued goes directly onto the Program EPROM. To terminate the **LOAD** command, the last characters to send are 2(two) hexFF characters. The hexFF characters tell the motor that it is the end of the file and to drop back into regular command mode.

Related Command:

LOCKP

RCKS

RUN

RUN?

UP

UPLOAD

Details on the downloadable file:

When you compile an SMS file with the SMI software, it creates an SMX file extension with the same name in the same directory. This is the file you need to download to the motor.

So basically here is what you should do:

Do an initial download of your program to the motor from SMI on some other machine. Issue the "**RCKS**" command. This is the "Report Checksum" command. It will respond with a string in the form of:

```
RCKS        000000 0000EB P
```

where the 000000 0000EB will be different than shown and represent a unique 2-byte checksum to any given program. The P at the end will be either a P (passed) or F (failed). Keep this number in your own program/PLC that will do the downloading.

1. Store the SMX file for downloading.

2. Store the string received from the **RCKS** command above as well.

3. Establish serial communications with the motor.

4. Issue **RCKS** command

5. If it does not match the stored checksum number: Open the smx file. Issue the **LOAD** command. Start sending down all characters in the smx file from beginning to end. When the last character is read from the file and sent to the motor then send2(two) hexFF characters to the motor.

6. Issue **RCKS** command again If it comas back with the stored string (with the "P" at the end) then the download was successful.

7. Issue "**RUN**" to see if it works as expected.

Reasons for unsuccessful download:

a. Noise on serial port

b. Loss of connection during download.

c. Failure to send the two hexFF's before power-down.

d. The SMX file as SMI compiled it was altered in some way.

Note: If you were to open an SMX file in NotePad to look at it and then save it, Notepad will automatically add carriage return characters at the end of each line it sees. The resultant file will not work. Each carriage return would have to be stripped back out prior to download. So do not alter the smx file in any way from how SMI generated it.

LOCKP

Prevent User Program Upload

Related Command:

UP

UPLOAD

APPLICATION:	User program execution control
DESCRIPTION:	Prevents effects of **UP** and **UPLOAD**
EXECUTION:	N/A
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

NOTE:

(For motors with a plug-in Memory Module)

Once ***LOCKP*** *has been invoked the Memory Module* ***EEPROM*** cannot be unlocked and the module must be replaced to return to an unlocked condition.

LOCKP modifies the contents of the header file portion of the downloaded Program in the motor's EEPROM to prevent the contents from being uploaded. That is, the commands **UP** and **UPLOAD** will not actually be able to upload the program body or contents. This does not prevent the downloading of another program.

It is suggested that the **LOCKP** command is used after program development and testing is complete.

LOCKP is intended as a serial command only. It should be issued from the terminal screen.

It should not be in the actual downloaded code.

Once **LOCKP** is issued, issuing **UP** or **UPLOAD** will no longer produce results.

LOOP

Return to WHILE Program Flow Control

Related Command:

BREAK

WHILE

APPLICATION:	Program execution control
DESCRIPTION:	Terminator for **WHILE** expression
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

LOOP is the statement terminator for the **WHILE** control block. Each **WHILE** must have one and only one corresponding **LOOP**. Each time **LOOP** is encountered, program execution branches back to re-evaluate the **WHILE expression**.

The **WHILE** (*expression)* . . **LOOP** control block creates a program loop that repeatedly executes for as long as the expression value is true or non zero. The **expression** is evaluated at the time **WHILE** is first encountered, and each time program execution is sent back to the **WHILE** by the corresponding terminating **LOOP** statement. If the **expression** value is zero or false, program execution continues on the line of code just below the **LOOP** command.

For version 4.00 and higher the **SMI** compiler encodes the **LOOP** (corresponding) **WHILE** program address location within the executable file. No **WHILE**/**GOSUB** return stack is used to carry out the proper execution of the **LOOP** statement. Thus **LOOP** executes the function equivalent of a **GOTO** without the need for declaring a program statement label. Simply restated: **WHILE expression** .. **LOOP** is functionally encoded as **Cx WHILE expression** . . **GOTOx**. This means that it is legal to jump into a **WHILE** control loop directly from an external program location.

LOOP is not a valid terminal command. It is only valid within a user program.

(Continued on next page.)

Related Command:

BREAK

WHILE

EXAMPLE:

```
b=1
WHILE b<5
     PRINT(#13,"b=",b)
     b=b+1
LOOP
PRINT(#13,"Exit Loop")
END
```

Output will be:

```
b=1
b=2
b=3
b=4
b=5
Exit Loop
```

Related Command:

BASE

CTR

G

MC2

MC4

MF1

MF2

MF4

MS

SIZE

APPLICATION:	Motion mode control
DESCRIPTION:	Request **CAM** mode
EXECUTION:	Buffered pending a **G**
CONDITIONAL TO:	**BASE=expression** and **SIZE=expression**
LIMITATIONS:	Requires external encoder signal source
REPORT COMMAND:	**RMODE**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT MODE:	**MP**
FIRMWARE VERSIONS:	ALL

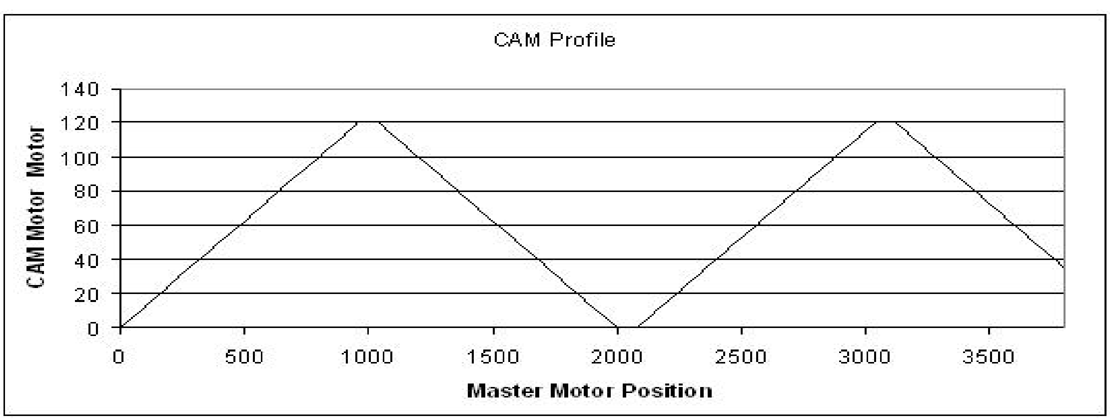
DETAILED DESCRIPTION:

MC puts the SmartMotor™ into **CAM Mode**, which causes the SmartMotor to follow a predetermined profile in accordance with an external encoder source. To set up a cam operation, you must also specify **BASE**, **SIZE**, **aw[0]..aw[SIZE]** position data and initialize to the external encoder counter. Start the camming motion by issuing a **G** command. The example below is a complete command sequence.

In **CAM Mode**, each value of the external encoder defines a required corresponding SmartMotor position; cams typically define a periodic motion profile or trajectory. **BASE** defines the number of encoder counts through which the external Cam moves before the required position mapping, or required motion, is exactly repeated.

EXAMPLE:

This is a "**saw tooth**" CAM with periodic motion of **BASE**=2000 external encoder counts and the motion interpolation divided into 25 (equal) segments:



(Continued on next page)

Related Command:

BASE

CTR

G

LOAD

MC2

MC4

MC8

MF1

MF2

MF4

MS

SIZE

```
'Example CAM MODE Setup:
BASE=2000   'Cam period
SIZE=25     'data segments (number of data points in table)
'CTR data interval = BASE/SIZE = 2000/25 = 80
'CAM motor will be at Data position every 80
'Master encoder counts:
'CTR=0, CTR=80, CTR=160,.... CTR=1840, CTR=1920, CTR=2000
'Now assigning data values beginning with aw[0]:
aw[0] 0 10 20 30 40 50 60 70 80 90 100.
aw[20] 110 120 120 110 100 90 80 70 60.
aw[19] 50 40 30 20 10 0.
MF4    'reset external encoder to zero
O=0    'reset internal encoder position
MC     'buffer CAM Mode
G      'start following the external encoder using cam data
```

The motor will now begin following the External (Master) encoder via the defined CAM profile above. The SmartMotor™ performs a practical cam application by partitioning the required cam trajectory definition into a number of linearly interpolated segments. The variable **SIZE** stores the number of segments. The segments are required to partition the **BASE** into a set of equally spaced intervals.

The set of required positions must always use the 16-Bit array values beginning at **aw[0]** and ending with **aw[SIZE]**. (aw[0 thru 99]). While this appears to limit the size of the cam table to 100 entries no larger than +32678, this is not the case. You can continually load new values into the **aw[ ]** array as the values get used **-** be sure you load the new values into **aw[ ]** array elements only after they have been used. The actual cam target positions can be increased by 2x, 4x or 8x with the **MC2**, **MC4** or **MC8** statements.

In other words, suppose aw[20]=100. If you use **MC2**, the effective value will be 200, with **MC4**, it will be 400, and with **MC8** it will be 800.

So **MC2**, **MC4** or **MC8** change the amplitude by a factor of 2X, 4X, or 8X respectively.

The **Cam Mode**, like any other position mode, is subject to the error band defined by the **E** value, and subject to limit switch inputs. While in motion during **Cam Mode**, flag **Bo** will be **0**, flag **Bt** will be **1** and flag **Be** will be **0**.

Note: PLS version Firmware allow the ability to run a relative CAM mode vice Absolute. Please consult the Firmware addendum documents for more detail.

Related Command:

BASE
CTR
G
MC
MC4
MC8
MF1
MF2
MF4
MS
SIZE

APPLICATION:	Motion mode control
DESCRIPTION:	Request **MODE CAM** with x2 multiplier
EXECUTION:	Buffered pending a **G**
CONDITIONAL TO:	**BASE=expression** and **SIZE=expression**
LIMITATIONS:	Requires external encoder signal source
REPORT COMMAND:	**RMODE**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT MODE:	**MP**
FIRMWARE VERSIONS:	Version 4.10 and higher

DETAILED DESCRIPTION: Same as mode **MC** in all regards with exception that all data points int he CAM table are multiplied by 2.

Suppose the following CAM table:

```
aw[0] 0 10 20 30 40 50 40 30 20 10 0.
```

The CAM motor would normally move through points 0, 10, 20, 30, etc....

But if **MC** is replaced with **MC2**, the CAM motor would instead mover though points 0, 20, 40, 60, 80, 100, 80, 60, 40, 20, and back to zero.

See the **MC** command for full details on CAM mode.

Related Command:

BASE
CTR
G
MC
MC2
MC8
MF1
MF2
MF4
MS
SIZE

APPLICATION:	Motion mode control
DESCRIPTION:	Request **MODE CAM** with x4 multiplier
EXECUTION:	Buffered pending a **G**
CONDITIONAL TO:	**BASE=expression** and **SIZE=expression**
LIMITATIONS:	Requires external encoder signal source
REPORT COMMAND:	**RMODE**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT MODE:	**MP**
FIRMWARE VERSIONS:	Version 4.10 and higher

DETAILED DESCRIPTION: Same as mode **MC** in all regards with exception that all data points in the CAM table are multiplied by 2.

Suppose the following CAM table:

```
aw[0] 0 10 20 30 40 50 40 30 20 10 0.
```

The CAM motor would normally move through points 0, 10, 20, 30, etc....

But if **MC** is replaced with **MC4**, the CAM motor would instead mover though points 0, 40, 80, 160, 340, 680, 340, 160, 80, 40, and back to zero.

See the **MC** command for full details on CAM mode.

Related Command:

BASE
CTR
G
MC
MC2
MC4
MF1
MF2
MF4
MS
SIZE

APPLICATION:	Motion mode control
DESCRIPTION:	Request **MODE CAM** with x8 multiplier
EXECUTION:	Buffered pending a **G**
CONDITIONAL TO:	**BASE=expression** and **SIZE=expression**
LIMITATIONS:	Requires external encoder signal source
REPORT COMMAND:	**RMODE**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT MODE:	**MP**
FIRMWARE VERSIONS:	Version 4.10 and higher

DETAILED DESCRIPTION: Same as mode **MC** in all regards with exception that all data points in the CAM table are multiplied by 8.

Suppose the following CAM table:

```
aw[0] 0 10 20 30 40 50 40 30 20 10 0.
```

The CAM motor would normally move through points 0, 10, 20, 30, etc....

But if **MC** is replaced with **MC8**, the CAM motor would instead mover though points 0, 80, 160, 240, 320, 400, 320, 240, 160, 80, and back to zero.

See the **MC** command for full details on CAM mode.

Related Command:

N/A

APPLICATION:	Motion mode control
DESCRIPTION:	Request MODE ANALOG AMPLIFIER
EXECUTION:	Immediate
CONDITIONAL TO:	Analog signal input available
LIMITATIONS:	N/A
REPORT COMMAND:	**RMODE**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT MODE:	MP
FIRMWARE VERSIONS:	4.15 and 4.40 series only

DETAILED DESCRIPTION:

MD50 converts the SmartMotor™ into a simple analog amplifier with motor. It accepts a 0 to 5V analog signal from I/O Port A pin with a 10-Bit A/D resolution. It is center weighted such that 2.5VDC gives zero PWM, 5VDC gives full positive PWM and 0VDC gives full negative PWM. Since Port A has a 5K pull-up resistor, if **MD50** is initiated with no connection to Port A, the motor will immediately be commanded to full positive PWM.

In operation, **MD50** is similar to **Mode Torque -** there is no trajectory calculation, so there is no position error associated with the resultant motion. Flags **Bo**, **Bt** and **Be** will all be zero. Motion is not affected by the **E** value. A motor in **MD50** mode responds to **RMODE** with **W**. **MD50** motion is conditional to limit switch input activity, (see **LIMD** , **LIMN**, **LIMH** and **LIML**), and **MD50** can be terminated with **OFF**, **S**, and **X**.

MD50, like **MT**, is immediate, and if the signal input at PIN A is a logical high or low, then full output will be requested instantly. If you assign Port A as an output, then set Port A to logic 1 or zero via UA=1 or UA=0 respectively, the motor will be commanded to full PWM in either positive or negative direction respectively.

MD50 performs an analog read on the I/O A pin signal every **PID** sample. A to D conversions are one of the most lengthy processes, so you may wish to use the **PID2** command if you are also running a user program that takes additional analog readings.

MD50 is closely tied to **MT.** When invoked, any prior value in the **"T"** parameter gets over written. To change from **MD50** to **MT**, be sure to first issue **OFF** and then **T=value** before issuing the **MT** command.

Related Command:

RCTR

CTR

MF1

MF2

MF4

APPLICATION: External encoder control

DESCRIPTION: Reset external encoder to zero

EXECUTION: Immediate

CONDITIONAL TO: External encoder inputs available

LIMITATIONS: N/A

REPORT COMMAND: **RCTR**

READ/WRITE: References read only external encoder **CTR**

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Resets **CTR** to zero

FIRMWARE VERSIONS: All except 4.40 series

DETAILED DESCRIPTION:

The command **MF0** allows the user to zero the second encoder register (see the **CTR** command) without changing the present motion mode of the SmartMotor™.

Following **MF0**, a secondary encoder signal, whether coming from an external source through the I/O A and B pins, will be continuously tracked and made available in the form of the **CTR** function; no gearing relationship is active, unless you write one yourself.

If the **Mode Follow with Ratio** (**MFR**) or the **CAM Mode** does not meet your requirement you can write your own loop and define a unique relationship between the incoming secondary encoder signal and the motor's position.

In addition, it may be that you do not want there to be any such relationship to motion. A common use of **MF0** is to take input from a quadrature output selector switch, especially in the context of a user interface, often including an LCD readout like the Animatics LCD2X20 and LCD4X20.

If the you are running in **MF**, **MFR**, **MC** or other encoder follow modes, be careful issuing **MF0** as the value of **CTR** is immediately zeroed. The SmartMotor will interpret this to be a sudden change in the master encoder input from its prior value to **0**.

Continued on next page

Related Command:

RCTR

CTR

MF1

MF2

MF4

EXAMPLE: (This example will print to the main channel)

```
b=4                          'b high for initial print
C1                           'Switch watch routine
      a=CTR&3                'a will recycle 0-3
      IF a!=b                'See if new a
            PRINT("SELECT: ",a,#13)
            b=a              'Update b, no re-prnt.
      ENDIF
      IF UGI==0              'Look for button
            GOSUB20          'Sub. to use a
      ENDIF
GOTO1                        'Infinite loop
```

Related Command:

CTR

MC

MC2

MC4

MC8

MF0

MF2

MF4

MS

APPLICATION: Motion mode control

DESCRIPTION: Mode Follow 4 external counts per 1 count of shaft motion

EXECUTION: Immediate

CONDITIONAL TO: External encoder inputs present

LIMITATIONS: Do not issue **MF0** while in mode **MF1**

REPORT COMMAND: **RMODE**

READ/WRITE: Associated external encoder is read only

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT MODE: **MP**

FIRMWARE VERSIONS: All except 4.40 series

DETAILED DESCRIPTION:

MF1 causes the SmartMotor™ to instantly and precisely follow a second, external, encoder signal from input pins A and B, resetting the external encoder **CTR** value to zero. For each 4 external encoder counts (in the same direction) received by the SmartMotor, the motor shaft will be requested to follow, moving 1 internal encoder count in the same direction. Velocity and acceleration feed-forward gains are not computed during this mode. Issuing any other mode such as **MT** or **MP** followed by **G** will take the SmartMotor™ out of this following behavior.

*For other ratios and fractional relationships see **Mode Follow with Ratio** (**MFR**)*

MF1 instantly turns on the servo and resets any position error. The servo off flag **Bo** is set to **0**, the trajectory flag **Bt** is set to **1**, and the position error flag **Be** is reset to **0**. The motion is restricted by the present **E** value. Issuing **E=0** will immediately cause a position error after 4 encoder counts, in the same direction, are received from the external encoder. The motion is also subject to the currently defined activity of the limit switches.

EXAMPLE:

```
MF1                'Reset CTR and Set follow mode
RMODE              'RESPONSE is "F"
WAIT=100000        'Follow for a while
MP                 'Revert to position mode
P=0                'Set destination for home
A=100              'Set acceleration
V=537*1000         'Set velocity
G            'Terminate following start position move
RMODE              'RESPONSE is "P"
```

Enable Mode-Follow Half-Quadrature

Related Command:

CTR

MC

MC2

MC4

MC8

MF0

MF1

MF4

MS

APPLICATION: Motion mode control

DESCRIPTION: Mode Follow 2 external counts per 1 count of shaft motion

EXECUTION: Immediate

CONDITIONAL TO: External encoder inputs present

LIMITATIONS: Do not issue **MF0** while in mode **MF2**

REPORT COMMAND: **RMODE**

READ/WRITE: Associated external encoder is read only

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT MODE: **MP**

FIRMWARE VERSIONS: All except 4.40 series

DETAILED DESCRIPTION:

MF2 causes the SmartMotor™ to instantly and precisely follow a second, external, encoder signal from input pins A and B, resetting the external encoder **CTR** value to zero. For each 4 external encoder counts (in the same direction) received by the SmartMotor, the motor shaft will be requested to follow, moving 1 internal encoder count in the same direction. Velocity and acceleration feed-forward gains are not computed during this mode. Issuing any other mode such as **MT** or **MP** followed by **G** will take the SmartMotor™ out of this following behavior.

For other ratios and fractional relationships see ***Mode Follow with Ratio*** *(**MFR**)*

MF2 instantly turns on the servo and resets any position error. The servo off flag **Bo** is set to **0**, the trajectory flag **Bt** is set to **1**, and the position error flag **Be** is reset to **0**. The motion is restricted by the present **E** value. Issuing **E=0** will immediately cause a position error after 4 encoder counts, in the same direction, are received from the external encoder. The motion is also subject to the currently defined activity of the limit switches

EXAMPLE:

```
MF2                'Reset CTR and Set follow mode
RMODE              'RESPONSE is "F"
WAIT=100000        'Follow for a while
MP                 'Revert to position mode
P=0                'Set destination for home
G            'Terminate following start position move
RMODE              'RESPONSE is "P"
```

MF4
Enable Mode Follow Full Quadrature

Related Command:

CTR

MC

MC2

MC4

MC8

MF0

MF1

MF2

MS

APPLICATION: Motion mode control

DESCRIPTION: Mode Follow 1 external counts per 1 count of shaft motion.

EXECUTION: Immediate

CONDITIONAL TO: External encoder inputs present

LIMITATIONS: Do not issue **MF0** while in mode **MF4**

REPORT COMMAND: **RMODE**

READ/WRITE: Associated external encoder is read only

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT MODE: **MP**

FIRMWARE VERSIONS: All except 4.40 series

DETAILED DESCRIPTION:

MF4 causes the SmartMotor™ to instantly and precisely follow a second, external, encoder signal from input pins A and B, resetting the external encoder **CTR** value to zero. For each 4 external encoder counts (in the same direction) received by the SmartMotor, the motor shaft will be requested to follow, moving 1 internal encoder count in the same direction. Velocity and acceleration feed-forward gains are not computed during this mode. Issuing any other mode such as **MT** or **MP** followed by **G** will take the SmartMotor™ out of this following behavior.

MF4 instantly turns on the servo and resets any position error. The servo off flag **Bo** is set to **0**, the trajectory flag **Bt** is set to **1**, and the position error flag **Be** is reset to **0**. The motion is restricted by the present **E** value. Issuing **E=0** will immediately cause a position error after 4 encoder counts, in the same direction, are received from the external encoder. The motion is also subject to the currently defined activity of the limit switches.

EXAMPLE:

```
MF4                 'Reset CTR and Set follow mode
RMODE               'RESPONSE is "F"
WAIT=100000         'Follow for a while
MP                  'Revert to position mode
A=100               'Set acceleration
V=537*1000          'Set velocity
P=0                 'Set destination for home
G             'Terminate following start position move
RMODE               'RESPONSE is "P"
```

Related Command:

Bd

CTR

D

G

MF1

MF2

MF4

MFR

MFMUL

V

APPLICATION: Mode follow control

DESCRIPTION: Mode follow external encoder with ratio **MFMUL/MFDIV**

EXECUTION: Buffered pending a **G**

CONDITIONAL TO: **D, MFMUL, MF1, MF2, MF4, V**

LIMITATIONS: Magnitude of ratio **MFMUL/MFDIV** must be less than 256

REPORT COMMAND: N/A

READ/WRITE: Write only

LANGUAGE ACCESS: Assignment, expressions and conditional testing

UNITS: Number

RANGE OF VALUES: -32768 to 332767

TYPICAL VALUES: **-5 < (MFMUL/MFDIV) < 5**

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

The ratio **MFMUL/MFDIV** specifies the gain for **Mode Follow with Ratio** (**MRF**). To use **MFR**, you will need to define the specific relationship (ratio) of the encoder count input to outgoing requested encoder counts of motion. The command **MFR** must be issued after both **MFMUL** and **MFDIV** have been set. Both **MFMUL** and **MFDIV** may positive or negative; use this fact to control the direction of shaft motion. Overly large ratio gains are flagged by the firmware setting the system flag **Bd**, and may be unstable. The error flag **Bd** will be set by **MFR** if the magnitude of **MFMUL/MFDIV** is **256** or greater. **MFR** does **NOT** reset **Bd** if already set by a prior procedure.

EXAMPLE:

```
Zd                  'reset Bd system flag
MF0                 'reset CTR
MFDIV=-10           'Denominator = -10
MFMUL=21            'Numerator = 21
MFR                 'Calculate Ratio, input 21 external counts
                    'resulting motion -10 counts
D=0                 'No phase shift
IF Bd GOTO12
ENDIF               'gain too large
G                   'Start Following
                    'Implementing Phase Adjust:
D=500               'Set Relative Distance
V=5000              'Set Relative Velocity
G                   'Start Phase Adjust
END
C12
      S             'Stop Motion
END
```

Related Command:

Bd

CTR

D

G

MF1

MF2

MF4

MFDIV

MFR

V

APPLICATION: Mode follow control

DESCRIPTION: Mode follow external encoder with ratio **MFMUL/MFDIV**

EXECUTION: Buffered pending a **G**

CONDITIONAL TO: **D, MFMUL, MF1, MF2, MF4, V**

LIMITATIONS: Magnitude of ratio **MFMUL/MFDIV** must be less than 256

REPORT COMMAND: N/A

READ/WRITE: Write only

LANGUAGE ACCESS: Assignment, expressions and conditional testing

UNITS: Number

RANGE OF VALUES: **-32768** to **332767**

TYPICAL VALUES: **-5** < **(MFMUL/MFDIV)** < **5**

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

The ratio **MFMUL/MFDIV** specifies the gain for **Mode Follow with Ratio** (**MRF**). To use **MFR**, you will need to define the specific relationship (ratio) of the encoder count input to outgoing requested encoder counts of motion. The command **MFR** must be issued after both **MFMUL** and **MFDIV** have been set. Both **MFMUL** and **MFDIV** may positive or negative; use this fact to control the direction of shaft motion. Overly large ratio gains are flagged by the firmware setting the system flag **Bd**, and may be unstable. The error flag **Bd** will be set by **MFR** if the magnitude of **MFMUL/MFDIV** is **256** or greater. **MFR** does **NOT** reset **Bd** if already set by a prior procedure.

EXAMPLE:

```
Zd            'reset Bd system flag
MF0           'reset CTR
MFDIV=-10     'Denominator = -10
MFMUL=21      'Numerator = 21
MFR           'Calculate Ratio, input 21 external counts
              'resulting motion -10 counts
D=0           'No phase shift
IF Bd GOTO12
ENDIF         'gain too large
G             'Start Following
              'Implementing Phase Adjust:
D=500         'Set Relative Distance
V=5000        'Set Relative Velocity
G             'Start Phase Adjust
END
C12
      S
END
```

Related Command:

CTR

D

G

MF1

MF2

MF4

MFDIV

MFMUL

V

APPLICATION: Motion mode control

DESCRIPTION: Request MODE FOLLOW WITH RATIO

EXECUTION: Buffered pending a **G**

CONDITIONAL TO: Ratio **MFMUL**/**MFDIV**, **D**, and **V**

LIMITATIONS: Magnitude of ratio **MFMUL**/**MFDIV** must be less than 256

REPORT COMMAND: **Ratio Cannot be reported**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: **-5** < **MFMUL**/**MFDIV** < **5 (non-reportable)**

DEFAULT MODE: **MP**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

The command **MFR** is used to implement a fractional relationship between an incoming secondary encoder signal and the SmartMotor™ internal shaft position, represented by the primary internal encoder count. The fractional relationship is defined the user set ratio of **MFMUL** to **MFDIV**.

To use **MFR**, you will need to define the specific desired relationship (ratio) of the external encoder input to shaft position, represented by the primary internal encoder count. The command **MFR** must be issued after both **MFMUL** and **MFMUL** have been specified. Both **MFMUL** and **MFDIV** may positive or negative; use this fact to control the resulting direction of shaft motion. Overly large ratio gains are flagged by the firmware setting the system flag **Bd**, and may be unstable. The error flag **Bd** will be set by **MFR** if the magnitude of **MFMUL**/**MFDIV** is **256** or greater. **MFR** does **NOT** reset **Bd** if already set by a prior procedure.

MFR followed by **G** will immediately turn on the servo and reset any position error. The servo off flag **Bo** is set to **0**, the trajectory flag **Bt** is set to **1**, and the position error flag **Be** is reset to **0**. The motion is restricted by the present **E** value. Issuing **E=0** would immediately cause a position error upon a single count of output motion being requested. The motion is also subject to the currently defined activity of the limit switches.

The fractional ratio is accurate to 23 binary places, this means that if the external encoder displacement during the motion exceeds **256*256*64** or **4,000,000** counts the **G** command should be reissued. Within this limitation, the calculated requested trajectory position is to within one count of mathematical precision.

Related Command:

CTR

D

G

MF1

MF2

MF4

MFDIV

MFMUL

V

Phase Offset Adjust:

In some applications, it may be necessary to introduce a phase shift to achieve proper alignment during **MFR** following.

To perform this shift, parameters **D** and **V** are employed to superimpose the corrective phase. During a phase shift **RD** will report the remaining phase difference.

EXAMPLE:

```
Zd                  'reset Bd system flag
MF0                 'reset CTR
MFDIV=-10           'Denominator = -10
MFMUL=21            'Numerator = 21
MFR                 'Calculate Ratio
                    'input 21 external counts
                    'resulting motion -10 counts
D=0                 'No phase shift
IF Bd GOTO12
ENDIF               'gain too large
G                   'Start Following
                    'Implementing Phase Adjust:
D=500               'Set Relative Distance
V=5000              'Set Relative Velocity
G                   'Start Phase Adjust
RMODE               'Response is "X"
END
C12
      S             'Stop Motion
END
```

Related Command:

A

D

E

G

MV

P

V

APPLICATION:	Motion mode control
DESCRIPTION:	Request MODE POSITION
EXECUTION:	Buffered pending a **G**
CONDITIONAL TO:	**A, D, E, G, P,V, PID loop**
LIMITATIONS:	Motor power sufficient to deliver acceleration **A** and velocity **V**
REPORT COMMAND:	**RMODE**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT MODE:	Default motion mode at power up
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The position mode is the default mode of the motor. If you ever change modes, you can return to position mode by issuing the **MP** command. The mode request is buffered until a **G** command is issued.

For a standard position mode move, the SmartMotor™ requires, at a minimum, a position, non-zero trajectory velocity **V** and an non-zero positive acceleration **A**. Position mode calculates the trajectory to the target position at the time the **G** command is issued. The preceding **P=expression** or **D=expression** determines if the move is to be absolute (destination target set equal to buffered **P** value) or relative (destination target set equal to current trajectory position plus the buffered **D** offset value). The **G** command may be issued at any time and may be repeated, particularly in the case of relative modes with **D=offset**.

*For a standard position mode move, the SmartMotor™ requires, at a minimum, a **P**osition, **V**elocity and an **A**cceleration.*

MP followed by **G** will immediately turn on the servo and reset any position error. The servo off flag **Bo** is set to **0**, the trajectory flag **Bt** is set to **1**, and the position error flag **Be** is reset to **0**. The motion is restricted by the present **E** value. Issuing **E=0** would immediately cause a position error upon a single count of output motion being required. The motion is also subject to the currently defined activity of the limit switches. **RMODE** will respond with a "**P**".

The SmartMotor performs trapezoidal and triangular velocity profiles by default, but because position, velocity and acceleration are all changeable "on the fly" (during a move), more elaborate profiles can be implemented through programming.

Continued on next page:

Related Command:

A

D

E

G

MV

P

V

Due to integer math truncation, **A** is effectively rounded down to the next even number. A value of **1** or **0,** therefore, produce a net acceleration of **ZERO**. In these instances, requests to change the current velocity produce no change in velocity until **A>=2** is requested and a new **G** command issued.

EXAMPLE:

```
MV                  'Velocity Mode
A=1000              'Set Acceleration
V=50000             'Set Velocity
G                   'Start Motion
WAIT=6000           'Wait 6000 samples
MP                  'Position Mode
A=50                'Set Acceleration
V=40000             'Set Velocity
P=1000              'Set Position
G                   'Start (change) Motion
WAIT=200            'Wait 200 samples
V=45000             'Change Velocity
P=0                 'Update Position
G                   'Start Motion
```

Related Command:

CTR

RCTR

RMODE

MFDIV

MFMUL

MSR

APPLICATION: Motion mode control

DESCRIPTION: Request MODE STEP AND DIRECTION

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: Step and direction input available

REPORT COMMAND: **RMODE**

READ/WRITE: Associated step and direction counter **CTR** is read only

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: **MS** resets **CTR** to zero

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

Opto-isolaton modules are suggested when using Step and Direction to assure reliable operation.

The **MS** command enables mode step and direction. In the step and direction mode the SmartMotor™ emulates a 2,000 or 4000, depending on model, step per revolution stepping motor and driver package, where I/O pins "A" and "B" are used to receive the step and direction inputs, respectively. In Step and Direction mode the SmartMotor is still operating in a closed loop fashion with the **PID** loop executing the servo functions, so tuning is still important.

The **MS** command is immediate and concurrently resets the external encoder **CTR** value to zero. For each external step pulse received by the SmartMotor, the motor will be requested to move one internal encoder count in the same direction as the direction input. For other ratios and fractional relationships see **Mode Follow with Ratio** (**MSR**). Velocity and acceleration parameters have no meaning in this mode. Issuing any other mode such as **MT** or **MP**, followed by **G**, will take the SmartMotor out of this following behavior.

Under **MS**, a logic level high on the **DIRECTION** input causes motion in the positive direction. That is, the shaft will move such that the internal encoder value will increase. The **STEP** input is enabled on the rising edge of the I/O **A** input signal and active while the signal is high. The actually motion of the step occurs on the signal falling edge. In accordance with standard rules, do not change the **DIRECTION** signal while the **STEP** signal is active (logic high). If you do, you can cause that step move to go the wrong direction.

Related Command:

CTR

RCTR

RMODE

MFDIV

MFMUL

MSR

MS will immediately turn on the servo and reset any position error. The servo off **Bo** is set to **0**, the trajectory flag **Bt** is set to **1**, and the position error flag **Be** is reset to **0**. The motion is restricted by the present **E** value. Issuing **E=0** would immediately cause a position error upon any encoder pulse being received from the external encoder. The motion is also subject to the currently defined activity of the limit switches.

As with most stepping systems, opto-isolation modules are suggested when using Step and Direction to assure robust operation.

EXAMPLE 1: IMMEDIATE MODE STEP, 1:1

```
MS                  'Reset CTR and step and direction mode
'Motor will immediately start following pulses at 1:1
RMODE               'RESPONSE is "S"
WAIT=100000         'Follow for a while
MP                  'Revert to position mode
P=0                 'Set destination for home
A=100               'Set acceleration
V=50000             'Set velocity
G                   'Terminate following start position move
RMODE               'RESPONSE is "P"
```

EXAMPLE 2: BUFFERED MODE STEP WITH RATIO OF 1:10

```
MS0           'Reset CTR to Zero, no motion will result
'This also sets up Port A and B
'for step and direction input mode
RMODE         'RESPONSE will be from previous mode!
MFMUL=10      'Multiply incoming pulses by 10
MFDIV=100     'Divide incoming pulses by 100
MSR           'Calculate Mode Step Ratio
G             'motor will now begin following a 1:10
RMODE         'RESPONSE is "X"
```

Related Command:

CTR

RCTR

MS

MSR

MFMUL

MFDIV

APPLICATION: Counter mode control

DESCRIPTION: Request step and direction counter mode

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: Step and direction input available

REPORT COMMAND: **RCTR**

READ/WRITE: step and direction counter **CTR** is read only

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: **MS0** resets **CTR** to zero

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The command **MS0** (**Mode Step** Zero) allows the user to zero the second encoder register (**CTR**) without changing the mode status of the SmartMotor™. Following MS0, incoming step and direction signals, using I/O pins A and B, will be fully decoded and presented in the form of the **CTR** variable; no gearing relationship is active, unless you write one yourself.

If the you are running in **MS MF**, **MSR**, **MFR**, **MC** or other encoder follow modes, be careful issuing **MS0** as the value of **CTR** is immediately zeroed. The SmartMotor will interpret this to be a sudden change in the master encoder input from its prior value to **0**.

As with most stepping systems, opto-isolation modules are suggested when using Step and Direction to assure robust operation.

EXAMPLE:

```
MS0          'reset CTR to zero
             'CTR value follows step and direction inputs
```

EXAMPLE:

It may be useful to monitor the quantity or frequency of incoming pulses.

```
a=CTR                 'Read CTR at start
WAIT=4069             'Wait one second
a=CTR-a               'Read the difference
PRINT("Rate=",a," Pulses/Sec")
```

MSR

Calculate/Enable Mode-Step-Ratio

Related Command:

Bd
CTR
D
G
MF1
MF2
MF4
MFDIV
MFMUL
V

APPLICATION: Motion mode control

DESCRIPTION: Request MODE STEP WITH RATIO

EXECUTION: Buffered pending a **G**

CONDITIONAL TO: Ratio **MFMUL**/**MFDIV**, **D**, and **V**

LIMITATIONS: Magnitude of ratio **MFMUL**/**MFDIV** must be less than 256

REPORT COMMAND: **RMODE**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: **-5** < **MFMUL**/**MFDIV** < **5**

DEFAULT MODE: **MP**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

MSR is used to implement a fractional relationship between an incoming secondary encoder signal and the SmartMotor™ internal shaft position, represented by the primary internal encoder count. The fractional relationship is defined the user set ratio of **MFMUL** to **MFDIV**.

To use **MSR**, you will need to define the specific relationship (ratio) of the external encoder input to shaft position, represented by the primary internal encoder count. The command **MSR** must be issued after both **MFMUL** and **MFDIV** have been specified. Both **MFMUL** and **MFDIV** may be positive or negative; use this fact to control the resulting direction of shaft motion. Overly large ratio gains are flagged by the firmware setting the system flag **Bd**, and may be unstable. The error flag **Bd** will be set by **MFR** if the magnitude of **MFMUL/MFDIV** is 256 or greater. **MFR** does **NOT** reset **Bd** if already set by a prior procedure.

MSR followed by **G** will immediately turn on the servo and reset any position error. The servo off **Bo** is set to **0**, the trajectory flag **Bt** is set to **1**, and the position error flag **Be** is reset to **0**. The motion is restricted by the present **E** value. Issuing **E=0** would immediately cause a position error upon a single count of output motion being required. The motion is also subject to the currently defined activity of the limit switches.

The fractional ratio is accurate to 23 binary places, this means that if the external encoder displacement during the motion exceeds **256*256*64** or 4,000,000 counts the **G** command should be reissued. Within this limitation, the calculated requested trajectory position is to within one count of mathematical precision.

In some applications, it may be necessary to introduce a phase shift to achieve proper

Related Command:

Bd

CTR

D

G

MF1

MF2

MF4

MFDIV

MFMUL

V

alignment during **MFR** following. To perform this shift, parameters **D** and **V** are employed to superimpose the corrective phase. During a phase shift **RD** will report the remaining phase difference.

As with most stepping systems, opto-isolation modules are suggested when using Step and Direction to assure robust operation.

EXAMPLE:

```
Zd                    'reset Bd system flag
MFDIV=-10             'Numerator = 21
MFMUL=21              'Numerator = 21
MSR                   'Calculate Ratio
                      'input 21 external counts
                      'resulting motion -10 counts
D=0                   'No phase shift
IF Bd GOTO5 ENDIF           'gain too large

G                     'Start Following
```

Implementing Phase Adjust:

```
D=500                 'Set Relative Distance
V=5000                'Set Relative Velocity
G                     'Start Phase Adjust
RMODE                 'RESPONSE  is "X"
C5
END
```

Related Command:

T=exp

APPLICATION:	Motion mode control
DESCRIPTION:	Request MODE TORQUE
EXECUTION:	Immediate
CONDITIONAL TO:	-1023 < **T** < 1023
LIMITATIONS:	None
REPORT COMMAND:	**RMODE, RT**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT MODE:	**MP**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

MT enables torque mode. In this mode, the motor is commanded to develop a specific **power level**, set by **T=expression**. **T** is in units of tenths of percent of the full capacity of the subject motor.

T=1023 results 100% PWM full torque in the positive direction.

T=-1023 results 100% PWM full torque in the negative direction.

The encoder still tracks position and can still be read with the **@P** variable, but the **PID** loop is off and the motor is not servoing or running a trajectory.

For any given torque and no applied load, there will be a velocity at which the back EMF of the motor will cause the acceleration to stop and the velocity to hold more or less constant. Under the no load condition, therefore, the **T** command will control velocity. As the delivered torque increases, the velocity decreases.

Note that this means that **MT** does not regulate torque. Instead, it delivers a fixed amount of power to the motor coils. As motor power is the product of torque and RPM, velocity decreases as the delivered torque increases and vice versa.

MT will immediately turn on the servo and reset any position error. The servo off flag **Bo** is set to **0**, the trajectory flag **Bt** is reset to **0**, and the position error flag **Be** is reset to **0**. The motion is not restricted by the present **E** value. Issuing **E=0** would have no effect upon the present motion. The motion is subject to the currently defined activity of the limit switches.

Related Command:

T=exp

Amplifier mode **MD50** effects the internal value of **T**.

The Reported value of T will not reflect the effect if switching from MD50 to MT mode. To change from mode **MD50** to mode **MT**, issue the sequence **OFF T=value MT**.

TORQUE MODE EXAMPLE:

```
UAI                 'Set I/O A as Input
T=0                 'Initialize T=0
MT                  'Enter Mode Torque
C1                  'Loop Forever
a=UAA-512   '2.5V = 0 Torque
                    'UAA will range from 0 to 1023 over
                    'an input voltage of 0 to 5VDC
T=2*a
GOTO1
END
```

The above example will track an incoming analog signal from 0 to 5 Volts **UAA= 0 to 1023**

Note: Do not attempt to regulate speed with Torque Mode. It is not designed for that and will give poor results. In like manner, it is difficult at best to attempt to place a speed limit on Torque mode. If the load decreases, the motor shaft speed will increase to a new equilibrium with th lighter load because Power must remain the same.

Related Command:

CTR

D

G

MF1

MF2

MF4

MFDIV

MFMUL

MT

T

V

APPLICATION: Motion mode control

DESCRIPTION: Dynamically brakes the motor

EXECUTION: Immediate

CONDITIONAL TO: **N/A**

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT MODE: N/A

FIRMWARE VERSIONS: >=4.76

DETAILED DESCRIPTION:

MTB places the SmartMotor™ into dynamic brake mode. In this mode, the motor coils are shorted together. Any motion of the shaft would normally produce Back EMF somewhat proportional to speed. Bt having the windings shorted out causes this Back EMF to be dissipated immediately. he result is a magnetic damping counter force to any attempted motion of the shaft for an external source.

IF MTB is issued while moving at a given speed, the shaft will come to a gradual stop at a rate proportional to the Back-EMF that was being generated at the time of issuing the MTB command. The shaft doesn't stop at any predetermined or commanded position and its trajectory is uncontrolled.

While in MTB, the motor will not produce any external DC bus voltage rise if the shaft is rotated because all windings are shorted back to themselves. As a result, the DC bus is protected against bus over voltage to within the drive stage current limits.

MTB is the default mode of operation for all motors with >=4.765 firmware. MTB is automatically issued any time the motor faults on over temp, position errors or travel limit crash.

The only mean to prevent this automatic action is to issue **BRKRLS** and **OFF** in that sequence,.

To Re-enable the automatic MTB function, issue **BRKSRV** (brake Servo)

Related Command:

A

D

E

G

MV

P

V

PID loop

APPLICATION: Motion mode control

DESCRIPTION: Request MODE VELOCITY

EXECUTION: Buffered pending a **G**

CONDITIONAL TO: **A**, **D**, **E**, **G**, **P**, **V**, **PID** loop

LIMITATIONS: Motor power sufficient to deliver **A**cceleration, **A**, and **V**elocity, **V**

REPORT COMMAND: **RMODE**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT MODE: **MP**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The **MV** command enables velocity mode. In velocity mode, the value of **V**, the target velocity, can be negative or positive. In contrast, position mode only uses the magnitude of the velocity parameter. Acceleration and velocity can be changed at any time, even during motion. The **G** command will initiate "on the fly" changes to any of the parameters.

If the actual velocity is greater that the value defined by **V**, then, upon reception of the next **G** command, the motor shaft will decelerate at the rate set by A until the excess velocity is removed. Conversely, if the actual velocity is less than **V** when the **G** command is entered, then the motor shaft motion will accelerate at the rate set by **A** until the requested velocity is attained. Similarly, if the actual velocity is in the opposite direction of **V** when the **G** command is entered, then the motor shaft motion will decelerate and then accelerate at the rate set by **A** until the requested velocity is attained.

Once the commanded velocity **V** is attained, motion continues at this rate, i.e. uniform velocity, indefinitely until the commanded velocity is changed or the mode is otherwise terminated. The encoder may wrap around during this mode, but no position error will be declared during the wrap.

In all firmware pror to 4.76, **MV** followed by **G** will immediately turn on the servo and reset any position error. The servo off **Bo** is set to **0**, the trajectory flag **Bt** is set to **1**, and the position error flag **Be** is reset to **0**. The motion is restricted by the present

Related Command:

A

D

E

G

MV

P

V

PID loop

E value. Issuing **E=0** would immediately cause a position error upon a single count of output motion being required The motion is subject to the currently defined activity of the limit switches. **RMODE** will respond with a **V**.

In firmware ==4.76 if ay prior errors exist, Zs r th appropriate command must be used to clear the associated error status bit flag.

Due to arcane digital math, **A** is effectively rounded down to the next even number. A values of **1** and **0** therefor produce a net acceleration of zero. In these instances, requests to change the current velocity produce no change in velocity until **A>=2** is requested and a new **G** command issued.

EXAMPLE:

```
    MV          'buffer velocity mode request
    A=2         'set the minimum possible buffered acceleration
    V=44444     'set buffered velocity
    G           'apply buffered motion parameter and mode
    WAIT=V      'do not use TWAIT since move is forever
    RMODE       'response is "V"
    V=-V        'prepare to reverse velocity direction
    A=2*A       'with double the present acceleration
    G           'reverse direction
    V=V/4       'prepare to slow to one quarter
                'of original velocity
    WAIT=V*V    'this is a valid expression
    G           'slow to one quarter original velocity
    WAIT=4096*10    'Wait 10 seconds
                    '(4069 servo samples = 1 second)
    X           'decelerate to stop at acceleration set by "A"
END
```

Related Command:

RP

MS0

MF0

APPLICATION:	Reset SmartMotor's™ encoder origin
DESCRIPTION:	Request SmartMotor's encoder origin change
EXECUTION:	Immediate
CONDITIONAL TO:	Present encoder count
LIMITATIONS:	SmartMotor's axis must be at rest
REPORT COMMAND:	**RP**
READ/WRITE:	Write only
LANGUAGE ACCESS:	Assignment
UNITS:	Encoder counts
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-2147483648** to **2147483647**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The **O=expression** allows the current position to be set to any value desired. You may declare the current position to be zero by entering **O=0** (the letter "**O**" the number zero). Similarly, you may declare the current position to be **1234** by entering **O=1234**. Using the **O=expression** does not modify previously entered **P** and **D** registers.

The **O=expression** avoids position drift and accumulated error by changing the SmartMotor's commanded position for the sample in which the command is executed, regardless of the real time position error and whether or not the shaft is moving. This command is useful in homing routines to set an origin or "home" position.

In firmware versions 4.12, 4.40 and later, The SmartMotor explicitly performs the **O=expression** operation before checking for excessive position error.

O=0 is often used to avert a 32 bit roll-over condition.

Continued on next page

Related Command:

RP

MS0

MF0

EXAMPLE: (reassigning origin does not destroy **P** and **P** buffered values)

```
A=20
V=100000
P=5000
MP
O=-1000     'present position set to negative 10000
GOSUB5
O=12345     'present position set to 12345
GOSUB5
D=5000
O=3000      'present position set to 3000
GOSUB5
END
C5
PRINT(#13,"Move origin is ",@P)
G
WHILE Bt LOOP
WAIT=4000
PRINT(#13,"Position is ")
RP
  RETURN
```

Program output is:

Move origin is	**-1000'**
Position is	**5000**
Move origin is	**12345**
Position is	**5000**
Move origin is	**3000**
Position is	**8000**

Related Command:

CCHN

RCHN

RCHN0

RCHN1

APPLICATION: Communication control

DESCRIPTION: Open a communications channel

EXECUTION: Immediate

CONDITIONAL TO: External communication i/o connections

LIMITATIONS: Hardware capabilities

REPORT COMMAND: **RCHN, RCHN0, RCHN1** report status conditions

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: See detailed description

TYPICAL VALUES: See detailed description

DEFAULT VALUE: **OCHN** (RS2, 0, N, 9600, 1, 8, C)

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

OCHN(TYPE,CHANNEL,PARITY,RATE,STOP BITS,DATA BITS, SPECIFICATION) opens a serial channel with the following specifications:

TYPE:	**RS2**, **RS4**, or **IIC**
CHANNEL:	0 (for host), 1
PARITY:	**O=odd**, **E=even**, **N=none**, **I=ignore**
Serial baud RATE:	2400, 4800, 9600, 19200, 38400 bps
AniLink bit RATE:	100 khz, 400 khz
STOP BITS:	1
DATA BITS:	8
Serial SPECIFICATION:	**C=cmd**, **D=data**
AniLink SPECIFICATION	**M=master**, **S=slave.**

Opening channel **0** as a **RS485** port dedicates I/O **G** to the RS485 control function, which is required for use with Animatics RS232 to RS485 converters like the RS485 and RS485-ISO. When using one of these adapters, you must ensure that the I/O G pin is configured as a TTL output with the **UGO** command before the channel is opened.

EXAMPLE:

```
OCHN(RS2,0,N,9600,1,8,C)     'performed at reset
```

Related Command:

G

MD50

MF1

MF2

MF4

MS

MT

MTB

APPLICATION: Motor control

DESCRIPTION: Turn servo off

EXECUTION: Next **PID** sample update

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMANDS: **RS** and **RBo**

READ/WRITE: Read only associated status flag, **Bo**

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: **OFF**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

OFF turns the power to the motor coils off and terminates the activity of the current motion mode. The system flag for Motor Off, **Bo,** will be set to **1**. The shaft will be free to coast to a stop, or to be rotated by other external means. The response to **RMODE** is **O** for off. The system flag, **Bt**, for trajectory in progress will be set to zero. The system position error flag, **Be**, to zero. The motor will still track any shaft movement and continue to update the present encoder position.

Note: In all firmware -4.76, the **OFF** command may result in switching to: **MTB** (Mode Torque Brake) depending on settings. If the otor is in default settings, **MTB** would be the default "Off-State mode when **OFF** is issued.

Please see **MTB** command for more details

P=expression

Set Commanded Absolute Position

Related Command:

@P
@PE
A
D
E
G
MP
V

APPLICATION: Trajectory control

DESCRIPTION: Set trajectory target position

EXECUTION: Buffered pending a **G** command

CONDITIONAL TO: **A**, **E**, **G**, **MP**, and **V**

LIMITATIONS: **A**, **V**, and **E** all non zero for real time position to change

REPORT COMMAND: **RP**

READ/WRITE: Read write

LANGUAGE ACCESS: Assignment, expressions, and conditional testing

UNITS: Encoder counts

RANGE OF VALUES: **-2147483648** to **2147483647**

TYPICAL VALUES: **-2147483648** to **2147483647**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

To specify an absolute target position to the SmartMotor's™ positional origin, set **P=target** position, positive or negative, and then follow with a **G** command.

P=expression sets the target position in **Position Mode**.

Unless a subsequent **D=expression** is issued, and as long as the appropriate trajectory parameters **A** and **V**, the motor will move to position specified by the last requested **P** value when the **G** command is issued.

The Mode of operation will be Absolute Positon Mode. The **RMODE** command will respond with "P"

RP will report the actual position, but if you set a variable equal to **P** such as **"a=P"**, that variable will be loaded with the last entered target position rather than the actual position. If you want to use the actual position in your program then use the **@P** variable such as **a=@P**.

CONTINUED ON NEXT PAGE:

Related Command:

@P

@PE

A

D

E

G

MP

V

EXAMPLE:

```
MP          'Change to position mode (default power-up mode)
P=1000      'Set buffered position to 1000 encoder counts
A=100       'Set acceleration
V=32212*50  'Set velocity
G           'Start Motion
TWAIT       'Wait for move to be performed
P=2000      'set a new buffered absolute target position
G
TWAIT
P=-2000     'Set a new (negative) buffered target position.
G
TWAIT
P=-1000
G
TWAIT
P=0
G
```

Related Command:

A

V

WAIT

CLK

APPLICATION:	**PID** sample rate control
DESCRIPTION:	Set **PID** sample rate to basic rate
EXECUTION:	Next **PID** update
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	PID Modulo samples
RANGE OF VALUES:	1, 2, 4, and 8 only.
TYPICAL VALUES:	N/A
DEFAULT RATE:	**PID1**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

The PID parameter sets the **PID** sample rate.

Valid values are: PID1, PID2, PID4, and PID8.

PID1 is the default. See the **RSP**, **Report Sample Period**, detailed description for determining the actual default sample rate frequency of your SmartMotor™. The default rate is close to 4000 samples/second.

Each **PID** sample period, the motor firmware scans and updates encoder position, trajectory generator, I/O, serial communications ports, and uses position error to perform the **PID** calculation to control the servo drive stage. The user program code, if any, is executed at any time the microprocessor is not involved in these activities.

The WAIT command is controlled by the system CLK (clock) The PID value changes the reported values to CLK and the effects of WAIT as well.

Both Velocity and Acceleration are impacted the same way the WAIt command is.

The values of 1, 2 4 and 8 mean the PID filter will react upon and update on position error to correct dive power every 1 2 4 or 8 PID samples. This does not change how code is executed but does change how much time is given to that execution. As a result, a program run at PID8 will typically run faster than a program run at PID1. However, since the frequency of PID updates to the drive stage are changed and samples of position error are done at different intervals, PID8 will result in a more course or abrasive motion than PID1. Special care should be taken when using the PID command due to this fact. Improper usage could result in very sporadic motion.

The next page show a comparison of the different PID values

Related Command:

A

V

WAIT

CLK

EXAMPLE:

```
'For a 2000 count encoder SmartMotor™:
'Using three fixed values under each of the PID settings
      v=128504           'use to Set commanded Velocity
      a=3167             'use to Set commanded Acceleration
      w=32552            'use to set Wait time

      PID1               'Default PID updates every servo sample
      WAIT=w             'Wait time     = 8    seconds
      V=v                'Velocity      = 2400 RPM
      A=a                'Acceleration = 400   RPS^2

      PID2               'PID updates   every 2 servo samples
      WAIT=w             'Wait time     = 4    seconds
      V=v                'Velocity      = 1200 RPM
      A=a                'Acceleration = 200   RPS^2

      PID4               'PID updates   every 4 servo samples
      WAIT=w             'Wait time     = 2    seconds
      V=v                'Velocity      = 600  RPM
      A=a                'Acceleration = 100   RPS^2

      PID8               'PID updates   every 8 servo samples
      WAIT=w             'Wait time     = 1    second
      V=v                'Velocity      = 300  RPM
      A=a                'Acceleration = 50    RPS^2

'
      PID1               'Return to Default PID
      WAIT=w             'Wait time    = 8 seconds

      END
```

As can be seen above, although the values used for Velocity, Acceleration, and Wait times remained the same, their effect was changed by a factor for the PID setting.

As a result, much care should be taken if changes are made in the middle of a program.

The PID parameter can be changed from PID1 to PID8 while the motor is sitting still to increase I/O scanning efficiency or other code execution and then returned to PID1 just prior to the next move. This is a technique used to increase response time for input triggers or mathematical calculations while there is no trajectory in progress.

PRINT( )

Print to Primary Communications Port

Related Command:

BAUD

CCHN

CMD

DAT

F=4

OHCN

PRINT1

PRINTA . .

. . PRINTH

APPLICATION:	Communications output control
DESCRIPTION:	Serial communications channel 0 **PRINT** function
EXECUTION:	Immediate, at present baud rate
CONDITIONAL TO:	Host or channel 0 serial port open
LIMITATIONS:	Output is not buffered, each character transmitted must wait for previous character to be finished. Next command not executed until entire **PRINT** function is done.
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	Values passed to PRINT string must be in the range of **-2147483648** to **2147483647**
TYPICAL VALUES:	Any of the ASCII character set
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION: PRINT ("ASCII string", #ascii_code, expression)

The **PRINT( )** command is used to transmit (output) data to the serial communications channel 0, RS232 TX and RS232 RX pins, otherwise known as the primary host channel. **PRINT( )** commands may be used to send output to a terminal for display, communicate with third party devices, or used to send commands to other motors.

WARNING:

*DO NOT USE A COMMENT MARKER (‘) WITHIN **PRINT( )**.*

IT WILL CAUSE A COMPILER ERROR

All items to be printed reside within the parentheses and are separated by commas. ASCII Text strings must be within double quotation marks. Variables are referenced by name and their ASCII string vales are printed. Simple math expressions are allowed.

Raw ASCII code values are prefixed by the # sign. The SPACE character is #32, TAB is #9, CARRIAGE RETURN is #13, and LINE FEED is #10.

PRINT( ) commands pause other code execution until the last character has been transmitted. No language commands, whether from the host or user program, are executed until the last character has been placed in the hardware transmit port.

What does this mean in practice? To put it more simply, there is a practical difference between **PRINT(a,b,c)** and the sequence **PRINT(a) PRINT(b) PRINT(c)**. Executing from within a program **PRINT(a,b,c)** will output the values of **a**, **b**, and **c** without the possibility of another command from the terminal interfering. Executing **PRINT(a) PRINT(b) PRINT(c)** from within a program while the host terminal is transmitting **GOSUB5** to the motor could lead to the execution sequence **GOSUB5**

Related Command:

BAUD

CCHN

CMD

DAT

F=4

OHCN

PRINT1

PRINTA..

..PRINTH

EXAMPLE:

```
OFF
KP=100       'Set Proportional Gain
O=1234       'Set origin to 1234
a=1 b=2
PRINT("Demonstration:",#13)
PRINT("a=",a)
PRINT(" and b=",b,#13)
PRINT("a+b=",a+b,#13)
PRINT("Position:",@P,#13)
WAIT=10                  'Allow time for serial buffer processing
PRINT("KP=",KP,#13)
PRINT("Hello World",#13,#13)
PRINT("Run Subroutines",#13)
WAIT=10
PRINT(#128,"GOSUB5 ",#13)     'tell all motors to run subroutine 5
WAIT=10
PRINT(#129,"GOSUB10",#13)     'Tell Motor-1 to run subroutine 10
WAIT=10
PRINT(#130,"GOSUB20",#13)     'Tell Motor-2 to run subroutine 20
WAIT=10
PRINT(#131,"GOSUB30",#13)     'Tell Motor-3 to run subroutine 30
x=123
PRINT(#132,"GOSUB",x,#13)     'Tell Motor-4 to run subroutine 123
v=100000
a=100
p=2000
PRINT(#130,"A=",a," V=",v,#13)      'Set speed and accel in motor 2
WAIT=10
PRINT(#130,"MP P=",p, " G",#13)     'Command Motor-2 to position
2000
WAIT=10
PRINT(#13,#13,"End of Demonstration.",#13)
END
----------------------------------------------------------
```

OUTPUT:

```
Demonstration:
a=1 and b=2
a+b=3
Position:1234
KP=100
Hello World

Run Subroutines
 GOSUB5
 GOSUB10
 GOSUB20
 GOSUB30
 GOSUB123
 A=100 V=100000
 MP P=2000 G


End of Demonstration.
```

PRINT1( )

Print to Secondary Communications Port

Related Command:

BAUD

CCHN

CMD

DAT

OCHN

PRINT

PRINTA . .

. . PRINTH

APPLICATION: Communications output control

DESCRIPTION: Serial communications channel 1 **PRINT** function

EXECUTION: Immediate, at present baud rate

CONDITIONAL TO: Channel 1 serial port open

LIMITATIONS: Output is not buffered. Each character transmitted must wait for previous character to be finished. Next command not executed until entire **PRINT** function is done.

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: Values passed to PRINT string must be in the range of **-2147483648** to **2147483647**

TYPICAL VALUES: Any of ASCII character set

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The **PRINT1( )** command is used to transmit (output) data to the serial communications channel 1, I/O pin E and F, otherwise known as the secondary serial channel. **Note: Proper OCHN command is required prior to use of PRINT1 !!**

All items to be printed reside within the parentheses and are separated by commas. ASCII Text strings must be within double quotation marks. Variables are referenced by name and their ASCII string vales are printed. Simple math expressions are allowed.

Raw ASCII code values are prefixed by the # sign. The SPACE character is #32, TAB is #9, CARRIAGE RETURN is #13, and LINE FEED is #10.

PRINT1( ) commands pause other code execution until the last character has been transmitted. No language commands, whether from the host or user program, are executed until the last character has been placed in the hardware transmit port.

What does this mean in practice? To put it more simply, there is a practical difference between **PRINT1(a,b,c)** and the sequence **PRINT1(a) PRINT(b) PRINT(c)**. Executing from within a program **PRINT1(a,b,c)** will output the values of **a**, **b**, and **c** without the possibility of another command from the terminal interfering. Executing **PRINT1(a) PRINT1(b) PRINT1(c)** from within a program while the host terminal is transmitting **GOSUB5** to the motor could lead to the execution sequence **GOSUB5**

Related Command:

BAUD

CCHN

CMD

DAT

OCHN

PRINT

PRINTA . .

. . PRINTH

EXAMPLE:

```
OFF
KP=100      'Set Proportional Gain
O=1234      'Set origin to 1234
a=1 b=2
PRINT1("Demonstration:",#13)
PRINT1("a=",a)
PRINT1(" and b=",b,#13)
PRINT1("a+b=",a+b,#13)
PRINT1("Position:",@P,#13)
WAIT=10                 'Allow time for serial buffer
processing
PRINT1("KP=",KP,#13)
PRINT1("Hello World",#13,#13)
PRINT1("Run Subroutines",#13)
WAIT=10
PRINT1(#128,"GOSUB5 ",#13)    'tell all motors to run
subroutine 5
WAIT=10
PRINT1(#129,"GOSUB10",#13)    'Tell Motor-1 to run subroutine
10
WAIT=10
PRINT1(#130,"GOSUB20",#13)    'Tell Motor-2 to run subroutine
20
WAIT=10
PRINT1(#131,"GOSUB30",#13)    'Tell Motor-3 to run subroutine
30
x=123
PRINT1(#132,"GOSUB",x,#13)    'Tell Motor-4 to run subroutine
123
v=100000
a=100
p=2000
PRINT1(#130,"A=",a," V=",v,#13)      'Set speed and accel in
motor 2
WAIT=10
PRINT1(#130,"MP P=",p, " G",#13)    'Command Motor-2 to
position 2000
WAIT=10
PRINT1(#13,#13,"End of Demonstration.",#13)
END
```

OUTPUT:

```
Demonstration:
a=1 and b=2
a+b=3
Position:1234
KP=100
Hello World

Run Subroutines
 GOSUB5
 GOSUB10
 GOSUB20
 GOSUB30
 GOSUB123
 A=100 V=100000
 MP P=2000 G


End of Demonstration.
```

PRINTA( ) . . . PRINTH( )

Print to External LCD Display

Related Command:

BAUD

CCHN

CMD

DAT

OCHN

PRINT

PRINT1

APPLICATION: Anilink communications output control

DESCRIPTION: Anilink communications **PRINT** function

EXECUTION: Immediate, at present baudrate

CONDITIONAL TO: Anilink LCD required for display

LIMITATIONS: Output is not buffered. Each character transmitted must wait for previous character to be finished. Next command not executed until entire **PRINT** function is done.

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: Expressions limited to **-2147483648** to **2147483647**

TYPICAL VALUES: Any of ASCII character set

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The **PRINTA( )** through **PRINTH( )** print to an LCD on the AniLink port or to a DIO-100 card. The command actually employs **DOUTA1** as the export mechanism. **PRINTA( )** outputs to an LCD that is addressed A, **PRINTB( )** to an LCD addressed B and so forth. As in the case with all AniLink expansion cards, the LCD address is selectable via jumpers

All items to be printed reside within the parentheses and are separated by commas. ASCII Text strings must be within double quotation marks. Variables are referenced by name and their ASCII string vales are printed. Simple math expressions are allowed.

Raw ASCII code values are prefixed by the # sign. The SPACE character is #32, TAB is #9, CARRIAGE RETURN is #13, and LINE FEED is #10.

There is a practical difference between **PRINTA(a,b,c)** and the sequence **PRINTA(a) PRINTA(b) PRINTA(c)**. Executing from within a program **PRINTA(a,b,c)** will be output the values of **a**, **b**, and **c** without the possibility of another command from the terminal interfering. Executing **PRINTA(a) PRINTA(b) PRINTA(c)** from within a program while the host terminal is transmitting **GOSUB5** to the motor could lead to the execution sequence **GOSUB5 PRINT(a) PRINTA(b) PRINTA(c)**, or **PRINTA(a) GOSUB5 PRINTA(b) PRINTA(c)** etc., depending upon the exact timing. The resulting output may or may not be the identical.

Related Command:

BAUD

CCHN

CMD

DAT

OCHN

PRINT

PRINT1

In **SMI**, the character " ‘ " is a comment delimiter. As such, if you put a " ‘ " inside of the **PRINT** statement, the **SMI** debugger will think that are commenting out the rest of the **PRINT** statement and flag it as an error. The SmartMotor™, however, doesn’t use comments, and will transmit the " ‘ " as a character. The easiest thing to do is simply not use " ‘ " within a print string.

PLEASE CONSULT MANUAL FOR LCD DISPLAY PRODUCTS FOR MORE ON THE FOLLOWING EXAMPLE.

EXAMPLE: (printing output to an AniLink LCD with port address A)

```
PRINTA(#56,#14,#6,#1)      '#56 initialize LCD,
                           #14 turns on cursor
                           #6 sets cursor
                              direction
                           #1 clears LCD and
                              resets position to
                              first character of
                              first line

PRINTA(#128,"I AM LCD ADDRESS A") 'Print stating
                           from character block
              128, far left character
        of first line

PRINTA(#192,"2nd. TEXT LINE")  'Print starting from
                      character block 192, far
        left character of second
   line of LCD

PRINTA(#148,"3rd. TEXT LINE")  'Print starting from
                      character block 148, 1st
        character 3rd line. Four
   line LCD4X20 only)

PRINTA(#212,"4th. TEXT LINE")  'Print starting from
                      character block 212, 1st
        character fourth line.
   Four line LCDX20 only
```

Q

Report Host-Mode Status

Related Command:

MD

APPLICATION: Report command

DESCRIPTION: Request HOST MODE status packet

EXECUTION: Immediate

CONDITIONAL TO: **MD** host mode

LIMITATIONS:

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Data packet - see detailed description

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

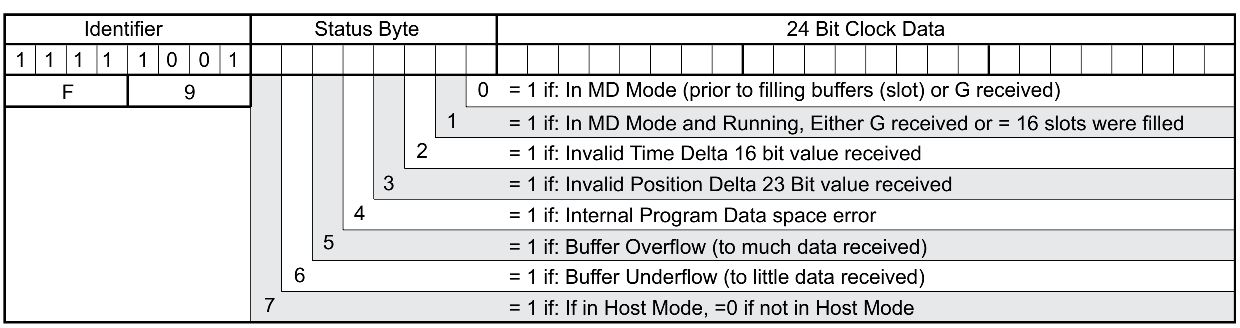
FIRMWARE VERSIONS: 4.15 and later. ??

DETAILED DESCRIPTION:

SEE SMI DOCUMENTATION FOR HOST UTILITY

Host Position Status Request Command **Q** Returns BINARY data only!

To track host positioning mode progress, the **Q** command returns status, clock, and space available in the dedicated circular buffer. The response to **Q** takes two forms, one while the mode not running and another while a trajectory is progress and no error has occurred. Both response conform to the overall byte format of 0xF9 + byte1 + byte2 + byte3 + byte 4 in binary. See diagram below:



A trajectory terminates if an unacceptable position error occurs, if invalid data received. if data overflow, or if data underflow. The host should send data pairs only when at least 3 empty data slots are available. **MD** responds to limit switches, trajectory will be aborted. **MD** mode uses **KV** feed forward for improved performance.

Related Command:

PRINT( )

APPLICATION:	Report command
DESCRIPTION:	Report user variable **a . . . z**
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Pre 4.00 only variables defined are **a, b, c, d, e, f, g** and **h**
REPORT VALUE:	**a** through **z**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Number
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-2147483648** to **2147483647**
DEFAULT VALUE:	**0**
RELATED COMMANDS:	N/A
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

*It is recommended that you use the alternative **"PRINT( )"** command when printing from your embedded programs because of its greater completeness and versitility.*

Ra reports the signed value of the variable **a** to the primary serial channel. A minus sign will precede negative values, no leading zeros are transmitted, and an ASCII carriage return terminates the transmitted data value. The equivalent **PRINT( )** command is **PRINT(a,#13)**. Use similar **PRINT** commands for **Rb**, **Rc**, through **Rx**, **Ry**, **Rz**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if **Ra** is received through channel 0, the response is sent through channel 0. If **Ra** is received through channel 1, the response goes out channel 1.

In SmartMotors so equipped, if **F=4** has been commanded, this report is redirected to serial channel 1 and the reported value is not be "seen" output the primary or currently active serial channel. Following **F=4**, the equivalent to **Ra** is **PRINT1(a,#13)**. **F=0** resets report commands to again be sent out the primary or currently active serial port.

Related Command:

PRINT( )

EXAMPLE:

```
F=0                     'use HOST channel
PRINT(#13,"F=0 ")
GOSUB5
F=4                     'redirect report output
PRINT(#13,"F=4 ")
GOSUB5
F=0                     'reset to default
END
C5
a=123
b=456
c=789
PRINT(a,b,c)
Ra
Rb
Rc
END
```

Host terminal only "sees" the following program output, Take note of the carriage returns (not explicitly shown here)

F=0 123456789123

456

789

Related Command:

N/A

APPLICATION:	Report command
DESCRIPTION:	Report user variable **aa**
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Not valid for pre 4.00 firmware
REPORT VALUE:	**aa**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Number
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-2147483648** to **2147483647**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

Raa reports the signed value of the variable aa to the primary serial channel. A minus sign will precede negative values, no leading zeros are transmitted, and an ASCII carriage return terminates the transmitted data value. The equivalent **PRINT( )** command is **PRINT(aa,#13)**. Use similar **PRINT** commands for **Rbb**, **Rcc**, through **Rxx**, **Ryy**, **Rzz**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if **Raa** is received through channel 0, the response is sent through channel 0. If **Raa** is received through channel 1, the response goes out channel 1.

In SmartMotors™ so equipped, if **F=4** has been commanded, this report is redirected to serial channel 1 and the reported value is not be "seen" output the primary or currently active serial channel. Following **F=4**, the equivalent to **Raa** is **PRINT1(aa,#13)**. **F=0** resets report commands to again be sent out the primary or currently active serial port.

Related Command:

N/A

EXAMPLE:

```
F         'use HOST channel
PRINT(#13,"F=0 ")
GOSUB5
F=4                 'redirect report output
PRINT(#13,"F=4 ")
GOSUB5
F=0                 'reset to default
END
C5
rr=123
ss=456
tt=789
PRINT(rr,ss,tt)
Rrr
Rss
Rtt
END
```

Host terminal only "sees" the following program output. Take note of the carriage returns (not explicitly shown here).

F=0 123456789123

456

789

Related Command:

N/A

APPLICATION:	Report command
DESCRIPTION:	Report user variable **aaa**
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Not valid for pre 4.00 firmware
REPORT VALUE:	**aaa**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Number
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-2147483648** to **2147483647**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

Raaa reports the signed value of the variable aaa to the primary serial channel. A minus sign will precede negative values, no leading zeros are transmitted, and an ASCII carriage return terminates the transmitted data value. The equivalent **PRINT( )** command is **PRINT(aaa,#13)**. Use similar **PRINT** commands for **Rbbb**, **Rcccc**, through **Rxxx**, **Ryyy**, **Rzzz**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if **Raaa** is received through channel 0, the response is sent through channel 0. If **Raaa** is received through channel 1, the response goes out channel 1.

In SmartMotors™ so equipped, if **F=4** has been commanded, this report is redirected to serial channel 1 and the reported value is not be "seen" output the primary or currently active serial channel. Following **F=4**, the equivalent to **Raaa** is **PRINT1(a,#13)**. **F=0** resets report commands to again be sent out the primary or currently active serial port.

Related Command:

N/A

EXAMPLE:

```
F=0        'use HOST channel
PRINT(#13,"F=0 ")
GOSUB5
F=4                 'redirect report output
PRINT(#13,"F=4 ")
GOSUB5
F=0                 'reset to default
END
C5
iii=123
jjj=456
kkk=789
PRINT(iii,jjj,kkk)
Rii
Rjj
Rkk
END
```

Host terminal only "sees" the following program output. Note the carriage returns (not explicitly shown here).

F=0 123456789123

456

789

Rab[index]

Report 8-Bit Array Data Value

Related Command:

N/A

APPLICATION:	Report command
DESCRIPTION:	Report user variable ab[index]
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Index range from 0 to 50
REPORT VALUE:	**ab[index]**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Number
RANGE OF VALUES:	**-128** to **127**
TYPICAL VALUES:	**-128** to **127**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

Rab[index] reports the signed value of the variable **ab[index]** to the primary serial channel. A minus sign will precede negative values, no leading zeros are transmitted, and an ASCII carriage return terminates the transmitted data value. The equivalent **PRINT( )** command is **PRINT(ab[index],#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if **Rab[23]** is received through channel 0, the response is sent through channel 0. If **Rab[23]** is received through channel 1, the response goes out channel 1.

The valid range of values of "index" is **0** to **200**. Index may be expressed directly as a number, a variable **a . . z**, the sum of two **a . . z** variables, or difference of two **a . . z** variables. There are no other combinations. See Example 1 for clarification; the example illustrates all legal index formats; thus **Rab[-6], Rab[t-6]**, and **Rab[-g]** do not represent valid index references. If you attempt to use a legal valid syntax, but the actual index value is out of range, system state flag **Bs** set to **1** and a syntax error message may be reported. See Examples 3 and 4.

The **ab[0]** to **ab[200]** variables represent signed 8 bit values; assignment of larger values is handled by truncating any extra leading data bits. The most significant bit is always considered to be a sign bit. See Example 2 for results when **ab[index]** is assigned a value larger than **255**.

Related Command:

N/A

EXAMPLE 1:

```
a=0                'assign test values
WHILE a<=6
      ab[a]=a
      a=a+1
LOOP
p=2 q=3 u=1 v=5
PRINT(ab[0]," ") Rab[0]       'report ab[0]
PRINT(ab[1]," ") Rab[1]       'report ab[1]
PRINT(ab[2]," ") Rab[p]       'report ab[2]
PRINT(ab[3]," ") Rab[q]       'report ab[3]
PRINT(ab[4]," ") Rab[v-u]     'report ab[4]
PRINT(ab[5]," ") Rab[v]       'report ab[5]
PRINT(ab[6]," ") Rab[v+u]     'report ab[6]
END
```

EXAMPLE 2:

```
a=254              'assign test values
WHILE a<=258
      i=a-252
      ab[i]=a      'assignment truncated to only 8 bits
      Rab[i]       'reported values are -2 -1 0 1 and 2
      a=a+1
LOOP
END
```

EXAMPLE 3:

```
Rab[201]     'sets Bs
             'fails to report a value but instead
             'emits a syntax error message
             'if syntax reports active
```

EXAMPLE 4:

```
v=605

Rab[v]       'sets Bs
             'fails to report a value but instead
             'emits a syntax error message

             'if syntax reports active
```

Ral[index]

Report 32-Bit Array Data Value

Related Command:

N/A

APPLICATION:	Report command
DESCRIPTION:	Report user variable **al[index]**
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Index range from **0** to **200**
REPORT VALUE:	**al[index]**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Number
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-2147483648** to **2147483647**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

Ral[index] reports the signed value of the variable **al[index]** to the primary serial channel. A minus sign will precede negative values, no leading zeros are transmitted, and an ASCII carriage return terminates the transmitted data value. The equivalent **PRINT( )** command is **PRINT(al[index],#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if **Ral[23]** is received through channel 0, the response is sent through channel 0. If **Ral[23]** is received through channel 1, the response goes out channel 1.

The valid range for the value of "index" is **0** to **50**. Index may be expressed directly as a number, a variable **a . . z**, the sum of two **a . . z** variables, or difference of two **a . . z** variables.

See Example 1 for clarification; the example illustrates ALL legal index formats; thus **Rab[-6]**,

Rab[t-6], and **Rab[-g] do not represent valid index references. If you attempt to use a legal valid syntax, but the actual index value is out of range, system state flag Bs set to 1** and a syntax error message may be reported See Examples 2 and 3.

The **al[0]** to **al[50]** variables represent signed 32 bit values; assignment of larger values is handled by truncating any extra leading data bits. The most significant bit, is always considered to be a sign bit.

Related Command:

N/A

EXAMPLE 1:

```
a=0                 'assign test values
WHILE a<=6
      al[a]=a
      a=a+1
LOOP
p=2 q=3 u=1 v=5
PRINT(al[0]," ") Ral[0]       'report al[0]
PRINT(al[1]," ") Ral[1]       'report al[1]
PRINT(al[2]," ") Ral[p]       'report al[2]
PRINT(al[3]," ") Ral[q]       'report al[3]
PRINT(al[4]," ") Ral[v-u]     'report al[4]
PRINT(al[5]," ") Ral[v]       'report al[5]
PRINT(al[6]," ") Ral[v+u]     'report al[6]

END
```

EXAMPLE 2:

```
Ral[51]             'sets Bs
                    'fails to report a value but instead
                    'emits a syntax error message
                    'if syntax reports active
```

EXAMPLE 3:

```
H=222
al[h]               'sets Bs
                    'fails to report a value but instead
                    'emits a syntax error message
                    'if syntax reports active
```

Raw[index]

Report 16-Bit Array Data Value

Related Command:

N/A

APPLICATION:	Report command
DESCRIPTION:	Report user variable aw[index]
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Index range from **0** to **100**
REPORT VALUE:	**aw[index]**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Number
RANGE OF VALUES:	**-32768** to **32767**
TYPICAL VALUES:	**-32768** to **32767**
DEFAULT VALUE:	**0**
RELATED COMMANDS:	N/A
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

Raw[index] reports the signed value of the variable **aw[index]** to the primary serial channel. A minus sign will precede negative values, no leading zeros are transmitted, and an ASCII carriage return terminates the transmitted data value. The equivalent **PRINT( )** command is **PRINT(aw[index],#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if **Raw[23]** is received through channel 0, the response is sent through channel 0. If **Raw[23]** is received through channel 1, the response goes out channel 1.

The valid range for the value of "index" is **0** to **100**. Index may be expressed directly as a number, a variable **a . . z**, the sum of two **a . . z** variables, or difference of two **a . . z** variables.

See Example 1 for clarification; the example illustrates ALL legal index formats; thus **Raw[-6], Raw[t-6]** and **Raw[-g]** do not represent valid index references. If you attempt to use a legal valid syntax, but the actual index value is out of range, system state flag **Bs** set to **1** and a syntax error message may be reported See Examples 3 and 4.

The **aw[0]** to **aw[100]** variables represent signed 16 bit values; assignment of larger values is handled by truncating any extra leading data bits. The most significant bit, is always considered to be a sign bit. See Example 2 for results when aw[index] is assigned a value larger than **256*256** or **65536**.

Related Command:

N/A

EXAMPLE 1:

```
a=0                'assign test values
WHILE a<=6
      aw[a]=a
      a=a+1
LOOP
p=2 q=3 u=1 v=5
PRINT(aw[0]," ") Raw[0]       'report aw[0]
PRINT(aw[1]," ") Raw[1]       'report aw[1]
PRINT(aw[2]," ") Raw[p]       'report aw[2]
PRINT(aw[3]," ") Raw[q]       'report aw[3]
PRINT(aw[4]," ") Raw[v-u]     'report aw[4]
PRINT(aw[5]," ") Raw[v]       'report aw[5]
PRINT(aw[6]," ") Raw[v+u]     'report aw[6]
END
```

EXAMPLE 2:

```
a=65534            'assign test values
WHILE a<=65538
      i=a-65534
      aw[i]=a      'assignment truncated to only 16 bits
      Rwb[i]       'reported values are -2 -1 0 1 and 2
      a=a+1
LOOP
END
```

EXAMPLE 3:

```
Raw[101]     'sets Bs
             'fails to report a value but instead
             'emits a syntax error message
             'if syntax reports active
```

EXAMPLE 4:

```
v=-605
aw[v]        'sets Bs
             'fails to report a value but instead
             'emits a syntax error message
             'if syntax reports active
```

Related Command:

N/A

APPLICATION:	Report command
DESCRIPTION:	Report buffered acceleration
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	**A**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Scaled encounter counts/**PID** sample/**PID** sample
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-2147483648** to **2147483647**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

RA reports the signed value of the buffered acceleration to the primary serial channel. A minus sign will precede negative values, no leading zeros are transmitted, and an ASCII carriage return terminates the transmitted data value. The equivalent **PRINT( )** command is **PRINT(A,[index],#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. Thus, if **RA** is received through channel 0, the response is transmitted through channel 0. If **RA** is received through channel 1, the response is transmitted through channel 1.

EXAMPLE:

```
V=3333
A=33
MV
G                       'use acceleration value 333
A=444
RA                      'returns the value 444
```

RAIN{port}{input}

Report Expanded Analog Input Value

Related Command:

AOUT

DIN

DOUT

APPLICATION:	Report command
DESCRIPTION:	Fetch and report Anilink peripheral analog input byte
EXECUTION:	Immediate IIC byte read, followed by transmit character
CONDITIONAL TO:	Port and input must exist
LIMITATIONS:	Port = A .. H and Input = 1, 2, 3, or 4
REPORT VALUE:	**AIN{port}{input}**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Unsigned numerical value
RANGE OF VALUES:	**0** to **255**
TYPICAL VALUES:	**0** to **255**
DEFAULT VALUE:	If requested input does not exist, the value **255** is returned
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

RAIN{address}{channel} fetches the unsigned 8 bit data value from the AIO-100 AniLink and reports it to the primary serial channel. The parameters address and channel refer to address and input channel, respectively, of the expansion card. No leading zeros are transmitted, and an ASCII carriage return terminates the transmitted data value. The equivalent **PRINT( )** command is **PRINT(AIN{address}{channel},#13)**.

Address may be A, B, C, D, E, F, G, or H, which is defined by jumper settings on the corresponding peripheral. The range of valid channels is 1 through 4.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. Thus, if the report command is received through channel 0, the response is transmitted through channel 0. If the report command is received through channel 1, the response is transmitted through channel 1.

EXAMPLES:

```
RAINC3          'valid port and channel
RAINA1          'valid port and channel
RAINW4          'invalid port, syntax error created
RAINB0          'invalid channel, syntax error created
```

RAMPS

Report Allowable PWM Limit

Related Command:

AMPS

T

MT

APPLICATION: Report command

DESCRIPTION: Report maximum allowed current to motor windings

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **AMPS**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: 1/1023 of maximum current permitted

RANGE OF VALUES: **0** to **1023**

TYPICAL VALUES: **1023**

DEFAULT VALUE: **1023**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

RAMPS reports the unsigned value of **AMPS**, the maximum power setting, to the primary serial channel. No leading zeros are transmitted, and an ASCII carriage return terminates the transmitted data value. The equivalent **PRINT( )** command is **PRINT(AMPS,#13)**

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. Thus, if the report command is received through channel 0, the response is transmitted through channel 0. If the report command is received through channel 1, the response is transmitted through channel 1.

EXAMPLE:

```
AMPS=333
RAMPS        'response is 333
AMPS=2000    'too large, entry auto corrected for safety
RAMPS        'response is 1023
```

Related Command:

Z

Za

ZS

APPLICATION: Report command

DESCRIPTION: Report system state over current latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **Ba**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: Over current event occurred

STATE VALUE 0: Over current has not occurred

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBa reports the value of the system over-current flag, **Ba**. It returns a **1** if an over-current has been detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Ba,#13)**

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. Thus, if the report command is received through channel 0, the response is transmitted through channel 0. If the report command is received through channel 1, the response is transmitted through channel 1.

EXAMPLE:

```
      PID1                'sample rate 4069 / second
      WHILE Bt            'report trajectory status about each second
            WAIT=4000
            PRINT(#13,"OVERCURRENT STATE      ")
                  RBa
            PRINT(#13,"OVERHEAT STATE         ")
                  RBh
            PRINT(#13,"POSITION ERROR STATE   ")
                  RBe
      LOOP
            PRINT(#13,"TRAJECTORY TERMINATED",#13)

END
```

Related Command:

RCHN

RCHN0

RCHN1

Zb

Z

ZS

APPLICATION: Report command

DESCRIPTION: Report system state flag communication parity error latched

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **Bb**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: Parity error event has occurred

STATE VALUE 0: Parity error event has not occurred

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBb reports the value of the communications parity error flag, **Bb**. It returns a **1** if any parity error has been detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bb,#13)**

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. Thus, if the report command is received through channel 0, the response is transmitted through channel 0. If the report command is received through channel 1, the response is transmitted through channel 1.

*Note:
A syntax error from the terminal causes **RCHN** to respond with value 4 but the value **CHN0** or **CHN1,** assigned to an expression is still zero.*

EXAMPLE:

```
C10         'communication status check subroutine
            'check both serial channel simultaneously
IF CHN0     'return immediately if no errors found
      PRINT("PARITY ERROR STATE     ") RBb
      PRINT("BUFFER OVERFLOW STATE  ") RBc
      PRINT("FRAMING ERROR STATE    ") RBf
      PRINT("SYNTAX ERROR STATE     ") RBs
ENDIF
RETURN
```

Related Command:

RCHN

RCHN0

RCHN1

Z

Zc

ZS

APPLICATION: Report command

DESCRIPTION: Report system state flag communication buffer overflow event latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **Bc**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: Communication buffer overflow event occurred

STATE VALUE 0: Communications buffer overflow has not occured

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBc reports the state of the serial communications overflow error flag, **Bc**. It returns a **1** if any overflow error has been detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT()** command is **PRINT(Bc,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. Thus, if the report command is received through channel 0, the response is transmitted through channel 0. If the report command is received through channel 1, the response is transmitted through channel 1.

Note:
*A syntax error from the terminal causes **RCHN** to respond with value 4 but the value **CHN** assigned to an expression is still zero.*

EXAMPLE:

```
C10         'communication status check subroutine
            'check both serial channel simultaneously
IF CHN0     'return immediately if no errors found
      PRINT("PARITY ERROR STATE     ") RBb
      PRINT("BUFFER OVERFLOW STATE  ") RBc
      PRINT("FRAMING ERROR STATE    ") RBf
      PRINT("SYNTAX ERROR STATE     ") RBs
ENDIF
RETURN
```

Related Command:

Z

Zd

ZS

APPLICATION:	Report command
DESCRIPTION:	Report system state flag math overflow event latch
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	**Bd**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Binary state
RANGE OF VALUES:	**0** to **1**
STATE VALUE 1:	Math overflow during product calculation or **MFMUL**/**MFDIV** division, has occurred
STATE VALUE 0:	No math overflow has occurred
FIRMWARE VERSIONS:	Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBd reports the value of the **MFMUL/MFDIV** math overflow error flag, **Bd**. It returns a **1** if any math overflow error was detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bd,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. Thus, if the report command is received through channel 0, the response is transmitted through channel 0. If the report command is received through channel 1, the response is transmitted through channel 1.

EXAMPLE 1:

```
Zd
RBd                'returns 0
a=1111111
b=2222222
c=a*b
Rc                 'returns -470886558
RBd                'returns 1
```

EXAMPLE 2:

```
Zd                 'reset  Bd
MFMUL=257          'initialize Mode Follow with Ratio
MFDIV=1
MFR
RBd                'returns 1 => MFR gain too large
```

If a standard 32 bit hand held calculator, in decimal mode, is used, it would also report an error.

RBe

Report Position Error Status Bit

Related Command:

G

RS

RW

RPW

Z

APPLICATION: Report command

DESCRIPTION: Report system state flag position error occurred latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **Be**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: Position error during trajectory motion occurred

STATE VALUE 0: No position error during trajectory has occurred

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBe reports the value of the position error flag, **Be**. It returns a 1 if a position error was detected and a 0 if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Be,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. Thus, if the report command is received through channel 0, the response is transmitted through channel 0. If the report command is received through channel 1, the response is transmitted through channel 1.

EXAMPLE:

```
O=0                  'Set current position to zero
A=100                'Set acceleration
V=50000              'Set velocity
P=100000000          'Set target position
E=1000               'default position error limit
MP                   'Set to position mode
G                    'Go and begin buffered move
WAIT=40000           'Wait abut 10 seconds
E=0                  'Force a position error by setting
                     'allowable limit to zero
WAIT=10              'Wait ten servo samples
RBe                  'response is 1
T=111
MT                   'position error reset by mode change
RPE                  'report position error limit,
                           response is 0
RBe                  'report position error bit,
                           response is 0
```

RBf

Report Communications Framing Error Status Bit

Related Command:

OCHN

RCHN

RCHN0

RCHN1

Z

Zf

ZS

APPLICATION: Report command

DESCRIPTION: Report system state flag communications framing error event latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **Bf**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: Parity error event occurred on either channel 0 or channel 1

STATE VALUE 0: No communication parity error event has occurred

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBf reports the value of the serial communications framing error flag, **Bf**. It returns a **1** if any framing error has been detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bf,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

EXAMPLE:

```
C10         'communication status check subroutine
            'check both serial channels simultaneously
IF CHN0     'return immediately if  no error found
      PRINT("PARITY ERROR STATE      ") RBb
      PRINT("BUFFER OVERFLOW STATE   ") RBc
      PRINT("FRAMING ERROR STATE     ") RBf
      PRINT("SYNTAX ERROR STATE      ") RBs
ENDIF
RETURN
```

*Note a syntax error from the terminal causes **RCHN** to respond with value 4 but the value **CHN** assigned to an expression is still zero.*

Report Over-Heat/RMS Over-Current Status Bit

Related Command:

TEMP

TH

THD

Z

APPLICATION:	Report command
DESCRIPTION:	Report real time system state motor overheat condition
EXECUTION:	Updated each **PID** sample
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	**Bh**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Binary state
RANGE OF VALUES:	**0** to **1**
STATE VALUE 1:	Motor in overheat condition
STATE VALUE 0:	Motor not is overheat condition
FIRMWARE VERSIONS:	Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBh reports the value of the overheat flag, **Bh**. It returns a **1** if an overheat was detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bh,#13)**

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

EXAMPLE:

```
WHILE Bt          'report trajectory  status
      WAIT=4000   'about once a second
      PRINT("OVER CURRENT STATE      ")
            RBa
      PRINT("OVER HEAT STATE      ")
            RBh
      PRINT("POSITION ERROR STATE   ")
      RBe
LOOP
PRINT(#13,"TRAJECTORY TERMINATED",#13)
```

Related Command:

Bx

I

RI

Z

APPLICATION: Report command

DESCRIPTION: Report system state flag index position latched

EXECUTION: Latch updated at **PID** sample if index event observed

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **Bi**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: Latched index encoder count reading available

STATE VALUE 0: No new latched index position available

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBi reports the value of the index available flag, **Bi**. It returns a **1** if a new index value was latched and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bi,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

Example: (Notice PRINT outputs from the following program)

```
A=10                'buffer a slow velocity mode move
V=4000
MV
E=100               'small error band
G                   'go
WHILE Bt
      RBi
      IF Bi
            PRINT("NEW INDEX VALUE ")
      ELSE
            PRINT("OLD INDEX VALUE ")
      ENDIF
      RI
      WAIT=400
LOOP
END
```

Related Command:

RCKS

RW

Z

APPLICATION: Report command

DESCRIPTION: Report EEPROM state flag I/O error event latch

EXECUTION: Immediate

CONDITIONAL TO: **RCKS**

LIMITATIONS: N/A

REPORT VALUE: **Bk**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: **RCKS** reported Program EEPROM checksum error

VST( ) reported Write Data EEPROM error

STATE VALUE 0: **RCKS** reported Program EEPROM checksum error

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBk reports the state of the checksum error flag, **Bk**. It returns a **1** if a checksum was detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bk,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

EXAMPLE:

```
RCKS
RBk   'reporting value, If 1 then the stored program is bad
```

Related Command:

Bm

RS

RW

S

Z

ZS

APPLICATION:	Report command
DESCRIPTION:	Report Left Limit State Latch
EXECUTION:	Updated each **PID** sample
CONDITIONAL TO:	**LIML**, **LIMH**, **UDM**
LIMITATIONS:	N/A
REPORT VALUE:	**BI**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Binary state
RANGE OF VALUES:	**0** to **1**
STATE VALUE 1:	Left limit switch has been active
STATE VALUE 0:	Left limit switch has not been active
FIRMWARE VERSIONS:	Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBI reports the value of the historical left limit flag, **BI**. It returns a **1** if an active left limit input was detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(BI,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

Report Historical Left-Over-Travel-Limit Status Bit

Related Command:

BI

Z

APPLICATION:	Report command
DESCRIPTION:	Report Historical Left Limit State
EXECUTION:	Updated each **PID** sample
CONDITIONAL TO:	**LINH**, **LIML**, **UDI**, **UDO**
LIMITATIONS:	N/A
REPORT VALUE:	**Bm**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Binary state
RANGE OF VALUES:	0 to 1
STATE VALUE 1:	Left limit switch active
STATE VALUE 0:	Left limit switch not active
FIRMWARE VERSIONS:	Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBm reports the value of the Historical left limit flag, **Bm**. It returns a **1** if an active left limit input was detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bm,#13)**

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

RBo

Report Motor-Off Status Bit

Related Command:

G

Z

ZS

APPLICATION: Report command

DESCRIPTION: Report real time system state motor off

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **Bo**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: Motor **PWM** signal is off

STATE VALUE 0: Motor **PWM** signal is on, motor coils are powered.

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBo reports the state of the motor off flag, **Bo**. It returns a **1** if an active left limit input was detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bo,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

EXAMPLE:

```
OFF
RBo                 'motor responds with a 1
T=100
MT                  'servo on, no PID loop
RBo                 'motor responds with a 0
MP
G                   'change mode, servo on with PID loop

RBo                 'motor still responds with a 0
OFF
RBo                 'motor responds with a 1
END
```

RBp

Report Historical Right-Over-Travel-Limit Logic State

Related Command:

Z

APPLICATION: Report command

DESCRIPTION: Report Historical Right Limit State

EXECUTION: Updated each **PID** sample

CONDITIONAL TO: **LIMH**, **LIML**, **UCI**, **UCP**, **UCO**

LIMITATIONS: N/A

REPORT VALUE: **Bp**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: Right limit switch active

STATE VALUE 0: Right limit switch not active

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBp reports the value of the Historical right limit flag, **Bp**. It returns a **1** if an active left limit input was detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bp,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

Related Command:

Z

APPLICATION:	Report command
DESCRIPTION:	Report Right Limit Active State Latch
EXECUTION:	Updated each **PID** sample
EXECUTION:	Updated each PID sample
CONDITIONAL TO:	**LIMH**, **LIMH**, **UCP**
LIMITATIONS:	N/A
REPORT VALUE:	**Br**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Binary state
RANGE OF VALUES:	**0** to **1**
STATE VALUE 1:	Right limit switch active
STATE VALUE 0:	Right limit switch not active
FIRMWARE VERSIONS:	Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBr reports the value of the real time right limit flag, **Br**. It returns a **1** if an active left limit input was detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Br,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

Related Command:

Z

Zs

ZS

APPLICATION: Report command

DESCRIPTION: Report system state flag scanning error event latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **Bs**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: Command scan error has occurred since **Bs** reset

STATE VALUE 0: Command scan error has not occurred since **Bs**
reset

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBs reports the value of the real time right limit flag, **Bs**. It returns a **1** if an active left limit input was detected and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bs,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. Thus, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

Scan errors result from malformed command and data syntax. An illegal array read/write access index also sets the scan error flag. Scan errors can occur from commands within program execution or received via either serial channel. A program encountering an illegal array access or syntax error should be carefully debugged. These programs may not execute accurately following the error.

Bs is reset by **ZS** and **Zs**.

NOTE: Downstream motors in a serial daisy chain will get their Bs bit set when upstream motors respond to report commands This is common and can be ignored.

EXAMPLE:

```
Zs          'reset any prior scan error state
j=88        'for use as array index
zzz=3333
al[j]=zzz   'value assigned is OK
            'but the index value is not, max
```

Array al[index] is location al[50]

```
RBs               'responds with 1
```

Related Command:

G

Z

APPLICATION: Report command

DESCRIPTION: Report real time system trajectory in progress state

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **Bt**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: Trajectory in progress

STATE VALUE 0: No trajectory in progress

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBt reports the state of the trajectory in progress flag, **Bt**. It returns a **1** if a a trajectory is in progress and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bt,#13)**

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

EXAMPLE:

```
OFF         'free shaft, no trajectory calculation
RBt         'motor responds with 0
A=555
V=777777
MV          'Set to Mode Velocity
G           'Start trajectory calculation
RBt         'motor responds with 1
WAIT=8000
T=7
MT          'Set to Mode Torque (no trajectory)
RBt         'motor responds with 0
WAIT=8000
OFF
WAIT=8000
MF4         'Mode Follow starts trajectory calculation
RBt         'motor responds with 1
END
```

Related Command:

Z

Zu

ZS

APPLICATION:	Report command
DESCRIPTION:	Report write array access error latch
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	**Bu**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Binary state
RANGE OF VALUES:	**0** to **1**
STATE VALUE 1:	Illegal report array value event occurred
STATE VALUE 0:	Illegal report array value event has not occurred
FIRMWARE VERSIONS:	Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBu reports the state of the array index error flag, **Bu**. It returns a **1** if there was any attempt to use an invalid index for an array variable and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bu,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

Bu is reset by **Z**, **ZS, and Zu**. Note, illegal array indexes always set **Bs** flag.

EXAMPLE: (if the following is executed by a user program)

```
ZS
m=44444
Raw[m]
PRINT(#13,"Issued Raw[illegal]",#13)
PRINT("Bu") Rbu   'Bu=1. array index range error occurred
PRINT("Bs") RBs   'Bs is 1, syntax occurred
PRINT(#13,"Issue ZS ",#13)
ZS
PRINT("Bu") RBu
PRINT("Bs",Bs) RBs
n=44444
s=aw[n]      'Illegal assignment behaves differently
PRINT(#13,"Assigned aw[illegal]",#13)
             'expression value is simply not assigned
PRINT("Bu") Rbu   'Bu is 0
PRINT("Bs") RBs   'Bs is 1
END
```

Related Command:

G

Z

APPLICATION: Report command

DESCRIPTION: Report system state flag

EXECUTION: Immediate

CONDITIONAL TO: Current motion mode

LIMITATIONS: N/A

REPORT VALUE: **Bw**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: Encoder wrap around occurred during a position move

STATE VALUE 0: Encoder wrap around event not recorded

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBw reports the state of the position wrap around flag, **Bw**. In any motion mode other than **MV**, **MT** or **MD50**, it returns a **1** if the encoder position wrapped and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bw,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

EXAMPLE: (try the follow Bw test program, at no instance is Bw set)

```
ZS
O=2147480000 'place close to wrap around at 2147483647
T=33
MT
PRINT(#13,"VALUE OF @ = ") RP
PRINT(#13,"VALUE OF Bw = ") RBw
WAIT=20000
IF @P<0
      PRINT(#13,"VALUE OF @ = ") RP
ENDIF
IF Bt
      PRINT(#13,"STILL GOING OK!")
ENDIF
PRINT(#13,"VALUE OF Bw = ") RBw
END
```

Related Command:

Bi

Z

APPLICATION: Report command

DESCRIPTION: Report real time index input state

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **Bx**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: **0** to **1**

STATE VALUE 1: Index input presently contacted

STATE VALUE 0: Index input not presently contacted

FIRMWARE VERSIONS: Versions 4.xx excluding HIRES Version 4.20

DETAILED DESCRIPTION:

RBx reports the state of the real time index flag, **Bx**. It returns a **1** if the current position is coincident with the encoder index **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(Bx,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

EXAMPLE: (Fast Index Find , Report Bx)

```
MP                'set buffered velocity mode
A=1000            'set fast acceleration
V=4000000         'set fast velocity
D=2100            'set relative distance just beyond
                  'one shaft turn
i=I               'clear and arm index capture
O=0               'force change to position register
G                 'start fast move
TWAIT             'wait till end of trajectory
P=I               'go back to index
G                 'start motion
TWAIT             'wait until end of trajectory
O=0               'set origin at index

RBx
```

Output will be 1

Related Command:

N/A

APPLICATION: Report command

DESCRIPTION: Report system state step direction change overrun event latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **By**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary state

RANGE OF VALUES: 0 to 1

STATE VALUE 1: Step direction overrun event occurred

STATE VALUE 0: Step direction overrun event has not occurred

FIRMWARE VERSIONS: 4.40 only!

DETAILED DESCRIPTION:

RBy reports the state of the step and direction overrun flag, **By**. It returns a **1** if the SmartMotor™ detected an invalid step, most likely due to an improper direction change, and a **0** if not. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(By,#13)**.

Note: IEEE standard states that the Direction bit should be looked at while the stp bit is low. If th direction bit transitions at the exact same time as the stp bit the By bit will be set.

Related Command:

CCHN

OCHN

RCHN0

RCHN1

APPLICATION:	Report command
DESCRIPTION:	Report serial communications status flags
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	Logical **OR** of **CHN0** with **CHN1**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Binary states
RANGE OF VALUES:	**0** to **15**
TYPICAL VALUES:	**0** to **15**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

RCHN returns the value of the historical communications function **CHN**. The read only function **CHN** holds binary coded historical error information about the two serial channels on the Smartmotor™. It gives the 4 bit status of either serial port channels 0 or 1, broken down as follows:

CHN bit **0** = **1** if either receive buffer has overflowed

CHN bit **1** = **1** if a framing error occurred on either channel

CHN bit **2** = **1** if a scan error occurred on either channel

CHN bit **3** = **1** if a parity error occurred on either channel

No leading zeros are transmitted, and it is followed by an ASCII carriage return. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(CHN,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

EXAMPLE:

```
RCHN        'test all command input  combined error flags
            'error occurred in value return is non zero
```

Related Command:

CCHN

OCHN

RCHN

RCHN1

APPLICATION: Report command

DESCRIPTION: Report serial communications channel 0 status flags

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **CHN0**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary states

RANGE OF VALUES: **0** to **15**

TYPICAL VALUES: **0** to **15**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

RCHN0 returns the value of the historical communications function **CHN0**. The read only function **CHN0** holds binary coded historical error information about the two serial channels on the SmartMotor™. It gives the 4 bit status of either serial port channels 0 or 1, broken down as follows:

CHN0 bit **0** = **1** if either receive buffer has overflowed

CHN0 bit **1** = **1** if a framing error occurred on either channel

CHN0 bit **2** = **1** if a scan error occurred on either channel

CHN0 bit **3** = **1** if a parity error occurred on either channel

No leading zeros are transmitted, and it is followed by an ASCII carriage return. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(CHN0,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

Related Command:

CCHN

OCHN

RCHN

RCHN1

EXAMPLE: (download and run the following)

```
END
C5                      'test individual flags
IF CHN0&4
      PRINT("CHANNEL 0 - scan error occurred")
ELSEIF CHN0&1
      PRINT("CHANNEL 0 - buffer overflow")
ENDIF
PRINT(#13)
RETURN
C10                     'test all flags
IF CHN0
      PRINT("CHANNEL 0 SERIAL ERROR !!")
ENDIF
PRINT(#13)
RETURN
```

Then from terminal type **RKK GOSUB5**.

Related Command:

CCHN

OCHN

RCHN0

RCHN1

APPLICATION: Report command

DESCRIPTION: Report serial communications channel 1 status flags

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **CHN1**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Binary states

RANGE OF VALUES: **0** to **15**

TYPICAL VALUES: **0** to **15**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

RCHN1 returns the value of the historical communications function **CHN1**. The read only function **CHN1** holds binary coded historical error information about the two serial channels on the SmartMotor™. It gives the 4 bit status of either serial port channels 0 or 1, broken down as follows:

CHN1 bit **0** = **1** if either receive buffer has overflowed

CHN1 bit **1** = **1** if a framing error occurred on either channel

CHN1 bit **2** = **1** if a scan error occurred on either channel

CHN1 bit **3** = **1** if a parity error occurred on either channel

No leading zeros are transmitted, and it is followed by an ASCII carriage return. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(CHN1,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

Related Command:

CCHN

OCHN

RCHN0

RCHN1

EXAMPLE: (download and run the following)

```
END
C5                      'test individual flags
IF CHN1&4
      PRINT1("CHANNEL 1 - scan error occurred")
ELSEIF CHN1&1
      PRINT1("CHANNEL 1 - buffer overflow")
ENDIF
PRINT1(#13)
RETURN
C10                     'test all flags
IF CHN1
      PRINT1("CHANNEL 1 SERIAL ERROR !!")
ENDIF
PRINT1(#13)
RETURN
```

Then from terminal type **RKK GOSUB5**

Related Command:

RCS1

APPLICATION:	Report command
DESCRIPTION:	Report channel 0 serial receive checksum
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	Checksum for channel 0 since prior **RCS**
LANGUAGE ACCESS:	N/A
READ/WRITE:	N/A
UNITS:	ASCII checksum number
RANGE OF VALUES:	**0** to **255**
TYPICAL VALUES:	**0** to **255**
DEFAULT VALUE:	Non zero
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

RCS reports the accumulated channel 0 checksum value to the primary serial channel. No leading zeros are transmitted, and it is followed by an ASCII carriage return. It is followed by an ASCII carriage return. There is no equivalent **PRINT( )** command.

The **RCS** checksum value is the simple 8 bit sum of all the ASCII bytes received by channel 0 serial channel. **RCS** resets the channel 0 checksum to zero after reporting the current value. See the ASCII Table in the appendix to map character to ASCII value. There is no **CS** command or function. It cannot be printed or assign to a variable. In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

EXAMPLE: (using the SMI terminal screen)

First noting	ASCII Space = 32	ASCII A = 65
	ASCII 1 = 49	ASCII C = 67
	ASCII 2 = 50	ASCII R = 82
	ASCII 3 = 51	ASCII S = 83
	ASCII "=" is 61 and SMI issues a space following a command	

```
Z
RCS          'response is 8 =  Mod 8
             '[82+67+83+32]=264-256=8
A=112
RCS          'response is 58 = Mod 8
             '[65+61+49+49+50+32+82+67+83+32]=570-512= 58
A=113
RCS          'response is 59, which is as expected,
             'one more than before.
```

RCS1

Report Secondary Serial Port Checksum

Related Command:

RCS

APPLICATION: Report command

DESCRIPTION: Report channel 1 serial receive checksum

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: Checksum for channel 0 since prior **RCS1**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Number

RANGE OF VALUES: **0** to **255**

TYPICAL VALUES: **0** to **255**

DEFAULT VALUE: Non zero

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

RCS1 reports the accumulated channel 1 checksum value to the primary serial channel. No leading zeros are transmitted, and it is followed by an ASCII carriage return. It is followed by an ASCII carriage return. There is no equivalent **PRINT( )** command.

There is no **CS1** command or function. You cannot print or assign a variable to **CS1**.

The **RCS1** checksum value is the simple 8 bit sum of all the ASCII bytes received by the channel 1 serial channel. **RCS1** resets the channel 1 checksum to zero after reporting the current value. See the ASCII Table appendice to map character to ASCII value.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

EXAMPLE: (see example RCS for additional explanation)

```
Z
RCS1            'response is 8 =  Mod 8
                '[82+67+83+32]=264-256=8
A=112
RCS             'response is 58 = Mod 8
                '[65+61+49+49+50+32+82+67+83+32]=570-512= 58
A=113
RCS1            'response is 59, which is as expected,
                'one more than before.
```

RCTR

Report Secondary Encoder Counter

Related Command:

CTR

ENC0

ENC1

MC

MF0

MFR

MS0

MSR

APPLICATION:	Report command
DESCRIPTION:	Report external encoder counter value
EXECUTION:	Updated each **PID** sample
CONDITIONAL TO:	External encoder signal available
LIMITATIONS:	N/A
REPORT VALUE:	**CTR**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Encoder counts or step pulses
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-2147483648** to **2147483647**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

RCTR reports the signed 32 bit value of the secondary encoder counter **CTR**. No leading zeros are transmitted and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(CTR,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

EXAMPLE:

```
MF0
RCTR                'responds with 0
```

Now provide external encoder input change.

```
RCTR                'response is non zero
MF4
RCTR                'CTR reset to zero
                    'response is 0
```

Report Commanded Relative Distance Value

Related Command:

A

E

G

P

MFR

MP

APPLICATION: Report command

DESCRIPTION: Report buffered relative move distance

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **D**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Encoder counts

RANGE OF VALUES: **-2147483648** to **2147483647**

TYPICAL VALUES: **-2147483648** to **2147483647**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

RD reports the value of the buffered relative move distance **D**. No leading zeroes are transmitted and it is followed by an ASCII carriage return. It is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(D,#13)**

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

EXAMPLE:

```
O=0                        'set up a move
MP
A=222
V=44444
D=-7777                    'first buffered D value to be used

G
D=2266                     'buffered D value
RD                         'response is 2266
```

RDIN{port}{channel}

Report Expanded Input Logic Status

Related Command:

DOUT

APPLICATION: Report command

DESCRIPTION: Fetch and report Anilink digital peripheral input byte

EXECUTION: Immediate byte read from IIC link

CONDITIONAL TO: Peripheral input attached to motor

LIMITATIONS: Returns **255** if port and channel does not exist

REPORT VALUE: **DIN{port}{channel}**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Number

RANGE OF VALUES: **0** to **255**

TYPICAL VALUES: **0** to **255**

DEFAULT VALUE: **255**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

RDIN{address}{channel} Report the unsigned 8 bit data value from the specified Anilink digital peripheral and reports it to the primary channel. The parameters address and channel refer to address and input channel, respectively, of the expansion card. No leading zeros are transmitted, and an ASCII carriage return terminates the transmitted data value. The equivalent **PRINT( )** command is **PRINT(DIN{address}{channel},#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

The command is most commonly used with an Animatics DIO-100 digital I/O module or an AniLink thumb wheel module.

Address may be A, B, C, D, E, F, G, or H, which is defined by jumper settings on the corresponding peripheral. The range of valid channels is 0 through 63, and is determined by the hardware.

EXAMPLE:

```
PRINT("DISPLAY THUMBWHEEL C INPUTS",#13,#13)
RDINC0                  'report wheel C, digit 0
RDINC1                  'report wheel C, digit 1
RDINC2                  'report wheel C, digit 2
```

EXAMPLE:

```
RDINK0                  'invalid port
RDINA66                 'invalid channel
RDINC
```

Report Maximum Allowable Position Error

Related Command:

A
E
G
P
MP
MV
V

APPLICATION: Report command

DESCRIPTION: Report maximum allowable position error

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **E**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Encoder counts

RANGE OF VALUES: -32768 to 32767

TYPICAL VALUES: -32768 to 32767

DEFAULT VALUE: 1000

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

RE reports the value of the allowable following error **E**. No leading zeros are transmitted and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(E,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

For normal operation **E** is greater than or equal to zero. If **E** is assigned a negative value a position error is immediately generated.

EXAMPLE:

```
A=554             'set up a buffered velocity move
V=666666
MV
E=300
G                 'go
WAIT=4000
RE                'response is 1000
E=-E
RE                'response is NOT -1000
```

I

RETURN

Return-From-Subroutine Program Flow Control

Related Command:

C

END

GOSUB

RUN

RUN?

APPLICATION: Program execution control

DESCRIPTION: Return subroutine execution to next program statement following present subroutine call

EXECUTION: Immediate

CONDITIONAL TO: A prior program statement **GOSUBn** was performed

LIMITATIONS: Prior to version 4.00 only total of 6 **WHILE** and **GOSUB** permitted at any one time. Version 4.00 supports up to 6 **GOSUBS** permitted at any one time.

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

Subroutines present a great opportunity to partition and organize your code.

The RETURN command is used to terminate a subroutine within a user program. Upon execution of the **RETURN**, program execution takes up immediately after the **GOSUB** that invoked the subroutine call. **RETURN** is normally executed from within the user program, but with care, the HOST terminal may also be used to issue a **RETURN** instruction.

The **RETURN** program locations are stored in memory called a stack. The stack depth is 6. Do not use more than 6 nested subroutines; if the the stack overflows, the program may will crash.

EXAMPLE:

```
PRINT("WAIT FOR HOST TERMINAL COMMANDS",#13)
GOSUB10                             'start of subroutine 10
PRINT("PROGRAM RECEIVED EXTERNAL RETURN")
END
C10                                 'start of subroutine 10
WHILE 1                             'wait for terminal commands
      WAIT=100                      'report terminal errors
      IF Bs
            PRINT(#13,"SCAN ERROR",#13)
            Zs
      ENDIF
LOOP
RETURN 'return to line just below GOSUB10 command
```

Report Last-Captured Index Pulse Location

Related Command:

Bi

Bx

I

Rbi

RBx

APPLICATION: Report command

DESCRIPTION: Report latched index position

EXECUTION: Immediate

CONDITIONAL TO: Index capture

LIMITATIONS: N/A

REPORT VALUE: **I**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Encoder counts

RANGE OF VALUES: **-2147483648** to **2147483647**

TYPICAL VALUES: **-2147483648** to **2147483647**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

RI reports the signed value of the latest captured index. No leading zeros are transmitted and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(I,#13)**.

If system flag **Bi** is **1** a "new" Index value is available. Issuing **RI** will reset **Bi** to zero.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

Example: (Notice PRINT outputs from the following program)

```
A=10                'buffer a slow velocity mode move
V=4000
MV
E=100               'small error band
G                   'go
WHILE Bt
      RBi
      IF Bi
            PRINT("NEW INDEX VALUE ")
      ELSE
            PRINT("OLD INDEX VALUE ")
      ENDIF
      RI
      WAIT=400
LOOP
END
```

Report Acceleration-Feed-Forward Gain Tuning Value

Related Command:

F

KA

KV

APPLICATION:	Report command
DESCRIPTION:	Report buffered acceleration feed forward gain
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	**KA**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	**PID** coefficient
RANGE OF VALUES:	**0** to **32767**
TYPICAL VALUES:	**0**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

RKA reports the signed value of the buffered **PID** acceleration feed forward gain value **KA**. No leading zeros are transmitted, and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(KA,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

The **KA** gain factor is only applied in position (**MP**) and velocity (**MV**) moves. Unlike the **KV** gain, the effectiveness of **KA** is difficult to verify. Future implementation will most likely be modified. The buffered **KA** value is not effective until a load filter command F is issued.

```
RKA                'Report present buffered KA
```

Related Command:

F

KI

KL

KP

APPLICATION:	Report command
DESCRIPTION:	Report buffered differential gain
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	**KD**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	**PID** coefficient
RANGE OF VALUES:	**0** to **32767**
TYPICAL VALUES:	**0**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

RKD reports the signed value of the buffered **PID** derivative gain value **KD**. No leading zeros are transmitted, and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(KD,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

```
RKD                 'Report present buffered KD
```

Report Gravitational Compensation Gain Tuning Value

Related Command:

F

KGON

KGOFF

APPLICATION:	Report command
DESCRIPTION:	Report buffered gravitational gain
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	**KD**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	**PID** coefficient
RANGE OF VALUES:	**-8388608** to **8388607**
TYPICAL VALUES:	**0**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

RKG reports the signed value of the buffered **PID** gravity constant **KG**. No leading zeros are transmitted, and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(KG,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

```
RKG                 'Report present buffered KG
```

Report Integral-Gain Tuning Value

Related Command:

F

KD

KI

KL

KP

APPLICATION: Report command

DESCRIPTION: Report buffered integral gain

EXECUTION: Immediate

CONDITIONAL TO: Integral limited by **KL** term

LIMITATIONS: N/A

REPORT VALUE: **KI**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: **PID** coefficient

RANGE OF VALUES: **0** to **32767**

TYPICAL VALUES: **0** to **20**

DEFAULT VALUE: Motor size dependant

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

RKI reports the signed value of the buffered **PID** integral gain value **KI**. No leading zeros are transmitted, and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(KI,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

```
RKI                 'Report present buffered KI
```

Related Command:

F

KD

KI

KL

KP

APPLICATION: Report command

DESCRIPTION: Report buffered proportional gain

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **KP**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: **PID** coefficient

RANGE OF VALUES: **0** to **32767**

TYPICAL VALUES: **40 to 400**

DEFAULT VALUE: Motor size dependent

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

RKP reports the signed value of the buffered **PID** proportional gain value ***KP***. No leading zeros are transmitted, and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(KP,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

```
RKP                 'Report present buffered KP
```

Related Command:

F

KD

KI

KL

KP

APPLICATION: Report command

DESCRIPTION: Report buffered inertial constant

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **KS**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: **PID** coefficient

RANGE OF VALUES: **0** to **255**

TYPICAL VALUES: **1**

DEFAULT VALUE: **1**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

RKS reports the signed value of the buffered **PID** sample rate modifier **KS**. No leading zeros are transmitted, and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(KS,#13)**. A value of **KS=0** is functionally equivalent to a **KS=1**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

```
RKS                'Report present buffered KS
```

Related Command:

F

KA

KV

APPLICATION: Report command

DESCRIPTION: Report buffered velocity feed forward gain

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **KV**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: **PID** coefficient

RANGE OF VALUES: **0 to 32767**

TYPICAL VALUES: **0** to **400**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

RKV reports the signed value of the buffered **PID** velocity feed forward value **KV**. No leading zeros are transmitted, and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(KV,#13)**

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

KV is very useful to fine tune long constant velocity trajectory profiles. Changes in **KV** are not updated until the load **PID** filter **F** command is issued.

```
RKV                 'Report present buffered KV
```

Related Command:

@P

@E

P

APPLICATION:	Report command
DESCRIPTION:	Report current position
EXECUTION:	Next **PID** sample
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	**@P**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Encoder counts
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-2147483648** to **2147483647**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

RP is the fundamental command to position data. **RP** reports the real time value of the primary encoder counter **@P**. No leading zeros are transmitted and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(@P,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

Do not confuse **RP** with **PRINT(P)**. **RP** returns the present position, whereas **PRINT(P)** returns the latest **P=expression** buffered requested absolute target position value. Notice also, **ENC1** changes the encoder position signal source from the default internal encoder to the external encoder inputs.

```
RP                'Report present shaft position
```

Related Command:

E

G

@PE

APPLICATION:	Report command
DESCRIPTION:	Report position error
EXECUTION:	Next **PID** sample
CONDITIONAL TO:	Servo active
LIMITATIONS:	Torque mode has zero position error
REPORT VALUE:	**@PE**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	Encoder counts
RANGE OF VALUES:	**-E** to **E**
TYPICAL VALUES:	**-E** to **E**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

RPE reports the signed value of the instantaneous position error **@PE**. No leading zeros are transmitted and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(@PE,#13)**.

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

Related Command:

E

G

@PE

EXAMPLE: (measure motion settling time)

```
O=0             'set current shaft position as origin
P=20000         'set target position
V=1000000       'set velocity
A=100           'set acceleration
G               'Go/start motion
WHILE Bt
LOOP               'wait for trajectory complete
a=CLK              'read the clock into variable
                   '"a". Clock measured in servo
                   'samples 4069 servo samples =1second.
GOSUB5             'observe settling motion
END
C5                             'subroutine label 5
IF @PE GOTO10 ENDIF            'de-bounce position error
IF @PE GOTO10 ENDIF
IF @PE GOTO10 ENDIF
IF @PE GOTO10 ENDIF
t=CLK-a            'Store clock into variable t
                   'measure settling time
PRINT(#13,"DECLARED AS SETTLED")
PRINT(#13,"SETTLING TIME ")
GOSUB20 PRINT(".")
GOSUB20 PRINT(" seconds")
RETURN
C10                                  'subroutine label 10
      PRINT(#13,"POSITION ERROR ")
      RPE                'report position error
GOTO5
C20                      'Subroutine label 20.
                         'perform    long divide
      s=t/4069
      PRINT(s)
      p=s*4069
      r=t-p
      t=10*r
RETURN

END
```

Related Command:

RPW

RW

APPLICATION:	Report command
DESCRIPTION:	Report motor status bits
EXECUTION:	Immediate
CONDITIONAL TO:	N//A
LIMITATIONS:	N/A
REPORT VALUE:	**S**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	8 motor status bits
RANGE OF VALUES:	**0** to **255**
TYPICAL VALUES:	**0** to **255**
DEFAULT VALUE:	**128** =- Motor OFF
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

RS reports the unsigned value of the present SmartMotor™ status byte **S**. No leading zeros are transmitted and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(S,#13)**. As does **RW**, **RS** resets the **Bh**, **Bl**, and **Br** flag values to zero.

A summary of **S**, the motor status byte, is:

Bo	Motor OFF	Status flag 7	
Bh	Excessive temperature	Status flag 6	reset by **RS**, **RW**
Be	Excessive position error	Status flag 5	
Bw	Encoder wrap around	Status flag 4	
Bi	Index report available	Status flag 3	reset by **RI**
Bl	Historical negative limit	Status flag 2	reset by **RW**, **RS**
Br	Historical positive limit	Status flag 1	reset by **RW**, **RS**
Bt	Trajectory in progress	Status flag 0	

In versions 4.15, 4.75, 4.41 and later, this has been changed to report through the current active serial channel and not just the primary port. That is, if the report command is received through channel 0, the response is sent through channel 0. If the report command is received through channel 1, the response goes out channel 1.

Related Command:

RPW

RW

Example:

```
O=10000                    'Set current shaft position
                           'as position 10000, set up move
P=0
A=222
V=33333
MP
G                          'go
WHILE Bt
      GOSUB5               'monitor for status change
LOOP
PRINT(#13,"FINAL REPORT",#13)
GOSUB5                     'final report
END
C5                         'subroutine 5
PRINT(#13,"STATUS BYTE VALUE ") RS
IF S&32     'logical AND status byte "S"
            'and position error status bit (0010 0000)
      PRINT(#13,"POSITION ERROR !!!")
ENDIF
IF S&16     'logical AND status byte "S"
            'and wraparound status bit    (0001 0000)
      PRINT(#13,"WRAP AROUND !!!")
ENDIF
IF S&1      'logical AND status byte "S"
            'and trajectory error status bit (0000 0001)
      PRINT(#13,"TRAJECTORY IN PROGRESS")
ENDIF
RETURN
```

Related Command:

CCHN

OCCHN

RS4

APPLICATION:	I/O Control
DESCRIPTION:	Restore PIN G I/O to default
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT MODE:	RS-232
FIRMWARE VERSIONS:	3.4x and higher

DETAILED DESCRIPTION:

The **RS2** puts the SmartMotor™ primary serial port into its default operating mode, RS232. The command is commonly used to put the primary serial port into RS232 mode after being previously put into RS485 mode with **RS4**. Among other things, **RS4** dedicates the I/O pin **G** to make the primary full-duplex RS232 channel a half-duplex RS485 channel. **RS2** frees the I/O **G** pin for general purpose use.

RS2 is also an argument in the **OCHN** command, used to put the target serial port in RS232 mode.

Related Command:

CCHN

ECHO

ECHO_OFF

OCCHN

RS2

APPLICATION:	I/O Control
DESCRIPTION:	PIN G is set to support RS485
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	**ECHO_OFF**
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT MODE:	RS232
FIRMWARE VERSIONS:	3.4x and higher

DETAILED DESCRIPTION:

The **RS4** command puts the primary serial port into RS485 mode. This allows you to use a RS232 to RS485 adapter, like the Animatics RS485 or RS485-ISO, on the primary serial port. As RS485 is half duplex and RS232 is full duplex, **RS4** dedicates the I/O pin **G** to control the direction of RS485 data. This is required for use with Animatics RS232 to RS485 converters like the RS485 and RS485-ISO. When using one of these adapters, you must ensure that the I/O **G** pin is configured as a **TTL** output with the **UGO** command before the channel is opened.

Note: **RS4** should only be used when the RS485ISO communications adapter is being used.

Report CPU speed and Firmware Revision

Related Command:

PID1

PID2

PID4

PID8

APPLICATION:	Report command
DESCRIPTION:	Report **PID** sample period and Firmware Revision
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT STRING:	ASCII alphanumeric string
READ/WRITE:	Read only
LANGUAGE ACCESS:	N/A
UNITS:	ASCII string
RANGE OF VALUES:	Firmware version dependant
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The report command **RSP** returns a five digit value of the **PID** sample period, followed by an ASCII string code representing firmware version. For versions 4.0 and higher, this basic sample rate is associated with the command **PID1**. The following is a table of firmware releases and **RSP** responses at the time of this printing:

The **PID** sample period, in microseconds, is the five digit number/100.

All version 4XX series motors respond in t form of:

24576/(firmware revision)

Example when sent to anSM2315D with 4.40c firmware.:

```
RSP                24576/440C
```

Related Command:

MT

APPLICATION: Report command

DESCRIPTION: Report torque request

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **T**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Encoder counts

RANGE OF VALUES: **-1023** to **1023**

TYPICAL VALUES: **-1000** to **1000**

DEFAULT VALUE: **1000**

FIRMWARE VERSIONS: <v4.95

DETAILED DESCRIPTION:

RT reports the value of the mode torque output value **T**. No leading zeros are transmitted and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(T,#13)**.

EXAMPLE: (this demonstrates the Severe Warning label in the margin)

```
      T=33                       'Test only with open shaft,
                           'setting torque value
      MT                         'set torque mode
      WAIT=4000                  'wait about 1 second
      PRINT("TORQE VALUE ")
      RT     'report torque requested
      MD50   'use analog voltage input to control torque
      'control mode. Potentiometer placed on I/O pin A.
'Voltage of 0V equates to t=-1023
            'and 5 V equates to T=1023
      WAIT=4000
      PRINT("TORQE VALUE ")   RT
      WAIT=4000
      MT                         'Effect: torque request of 33
                                 'has been destroyed
      PRINT("ISSUED MT")
      WAIT=4000
      T=33
```

SEVERE WARNING:

*If **MT** follows **MD50**, issue **OFF** and **T=expression** before the **MT** command.*

Related Command:

END

RUN?

APPLICATION: Program execution control

DESCRIPTION: Execute user EPPROM program beginning at initial command

EXECUTION: Immediate

CONDITIONAL TO: No effect if no EEPROM program exists

LIMITATIONS: Valid EEPROM stored program commands

REPORT COMMAND: **UP** and **UPLOAD**

READ/WRITE: EEPROM source

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT: **RUN** at power recycle, or software reset

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The **RUN** command will start a stored (downloaded) user EEPROM program.

Issuing a **RUN** command does not reset any motion, variable or I/O states.

It does reset the program execution pointer (Stack Pointer) to zero, and resets the internal **GOSUB** stack.

To test your program with a truly "fresh" start use the **Z** command to completely reset the motor as if it were newly powered up.

If a program exists within the SmartMotor™ user EEPROM it will automatically be run every time the motor is turned on.

To prevent this, make **RUN?** the first program statement of your user program, or if you wish, place **RUN?** anywhere in your program. Upon encountering a **RUN?** the program interpreter, execution machine, recalls whether or not the RUN command was previously issued, and if **RUN** was not issued, program execution ceases. This is similar to to encountering an **END** statement, except that a subsequent **RUN** command causes the program to take up after the **RUN?** statement.

Version 4 SmartMotors provide an abort facility to prevent auto-execution of stored program. In version 4.0, 4.10 through 4.13 and 4.2 SmartMotors, the stored program is aborted if any recognizable serial character is received during the first 500 mil-

Related Command:

END

RUN?

liseconds after power up or reset. In versions 4.15, 4.75 and onwards, the stored program is aborted if the serial character string "EE", or subset "EE" of "EEEEEEEEEEEE ...." during the first 500 milliseconds after power up or reset.

EXAMPLE: (user program with possible halt)

```
PRINT(" LOADING TRAJECTORY")
A=100
V=1000000
P=1000000
MP
PRINT(" Type RUN to start",#13      'Prompt user for
                  "RUN" command
RUN?        'Run command requested. Stop program
            'execution until "RUN" command is received.
PRINT(" EXECUTING TRAJECTORY")
G
END
```

Halt Program Execution until RUN Received

Related Command:

END

RUN

APPLICATION: Program execution control

DESCRIPTION: Halt execution of user program commenced without **RUN**

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: Valid via serial communication or program read

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT: Halts programs automatically started at power up

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

If a program exists within the SmartMotor™ user EEPROM it will automatically run every time the motor is turned on. To prevent this make **RUN?** the first program statement of the user program, or place **RUN?** anywhere in the program. When **RUN?** is encountered the program interpreter, execution machine, recalls whether or not the **RUN** command was previously issued, and if **RUN** was not issued, program execution ceases. This is similar to to encountering an **END** statement, except that a subsequent **RUN** command causes the program to take up after the **RUN?** statement.

The program will only begin when explicitly told to run by a "RUN" command sent by a host.

RUN? does not terminate the present motion mode or trajectory, change motion parameters such as **E**, **A**, **V**, and **KP**, or alter the present value of the user variables.

RUN? may be issued externally through the serial channel. It can distinguish motors which have suffered a power reset or software reset **Z** from those motors in a daisy chain which have not performed a reset..

EXAMPLE:

```
GOSUB1          'always execute subroutine 1 upon any reset
GOSUB2          'always execute subroutine 2 upon any reset

PRINT("Type RUN to start",#13)       'Prompt user for
                                     'RUN command

RUN?            'Halt program execution until
                'RUN command is received

GOSUB3          'conditionally execute subroutine 3
END
```

Related Command:

@V

V

APPLICATION: Report command

DESCRIPTION: Report current velocity

EXECUTION: Next **PID** sample

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT VALUE: **@V**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: Scaled encoder counts/sample

RANGE OF VALUES: **-2147483648** to **2147483647**

TYPICAL VALUES: **-3200000** to **3200000**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

RV reports the signed 32 bit value of the current trajectory velocity **@V**. It is not the actual velocity, but what the velocity is supposed to be at the time the **RV** command was executed. No leading zeros are transmitted and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(@V,#13)**.

EXAMPLE: (monitor acceleration ramp)

```
O=0                 'set up a velocity move
E=4000
A=10
v=4000000
V=v
MV
G
WHILE @V<v          'monitor velocity while
      IF Be               'accelerating
            BREAK         'exit if position error
      ENDIF
      GOSUB5              'report trajectory velocity
LOOP
GOSUB5                    'final report
END
C5
PRINT(" VELOCITY ")
RV                        'report trajectory
WAIT=4000                 'commanded velocity request
RETURN
```

Related Command:

RPW

RW

Whoops, some more of those pesky asterisks that don't seem to go anywhere

APPLICATION:	Report command
DESCRIPTION:	Report extended motor status flags
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT VALUE:	N/A
READ/WRITE:	Report only
LANGUAGE ACCESS:	None
UNITS:	16 motor status bits
RANGE OF VALUES:	**
TYPICAL VALUES:	**
DEFAULT VALUE:	128 = Motor OFF
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

RW reports the unsigned value of the present SmartMotor™ status word **W**. No leading zeros are transmitted and it is followed by an ASCII carriage return. The equivalent **PRINT( )** command is **PRINT(W,#13)**. As does **RS**, **RW** resets the **Bh**, **Bl**, and **Br** flag values to zero.

A summary of **W**, the motor status word, is:

Bk	EEPROM checksum failure	bit 15	
Ba	AMPS over current latch	bit 14	
Bs	Syntax error	bit 13	
Bu	Array index range error	bit 12	
Bd	Math overflow error	bit 11	
Bm	Real time negative limit active	bit 10	
Bp	Real time positive limit active	bit 9	
Bx	Real time index report	bit 8	
Bo	Motor OFF	bit 7	
Bh	Excessive temperature	bit 6	reset by **RPW**, **RW**, **RS**
Be	Excessive Position error	bit 5	
Bw	Position wrap around	bit 4	
Bi	Historical index report latched	bit 3	reset by **RI**, bit 3
Bl	Historical negative limit	bit 2	reset by **RPW**, **RW**, **RS**
Br	Historical positive limit	bit 1	reset by **RPW**, **RW**, **RS**
Bt	Trajectory in progress	bit 0	

If **RW** is reported the historical limit and overheat flags are immediately reset after the request command operation is completed. The value **W** cannot be assigned to a variable.

S (as command)

Stop Motion Quickly

Related Command:

A

D

E

G

MP

MV

P

X

APPLICATION: Motion mode control

DESCRIPTION: Abruptly stop motor motion

EXECUTION: Immediate

CONDITIONAL TO: **E** value

LIMITATIONS: If position error exceeds **E**, motor will shut off and coast to a stop

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

CAUTION

*Careful use of the **S** command is vital.*

The **S** command causes an emergency stop. It does not turn the motor off, rather it sets the target position at the current position. The resulting commanded motion will be very abrupt. In some cases it will be so abrupt that the amplifier can over current or the servo error can exceed the maximum allowable error set by the **E** command. This will, in turn, cause the motor to be turned off and coast. Consequently, careful use of the **S** command is vital. Following **S**, the motion mode is position mode, unless a position error is created, regardless of the mode it was in before. The response to **RMODE** will be "R." If the motion that was stopped was a Mode Position move, the previous target **P** or **D** values are still retained.

EXAMPLE:

```
A=100
V=1000000
P=5000000
G
WHILE Bt
IF UAI              'E-stop if PIN A high
S                         'Stop Abruptly
PRINT("Emergency Stop")
ENDIF
LOOP
```

S (as status byte)

8-Bit System Status Byte

Related Command:

RPW

RS

RW

APPLICATION:	Program execution control
DESCRIPTION:	Fetch primary motor status flags
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	**RS**
READ/WRITE:	Read only
LANGUAGE ACCESS:	Expressions and conditional testing
UNITS:	Status byte
RANGE OF VALUES:	**0** to **255**
TYPICAL VALUES:	**0** to **255**
DEFAULT VALUE:	**128**= Motor OFF
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

S is the value of the primary motor status byte, composed of 8 system flags states. The individual meaning of each flag is as follows:

Bo	Motor OFF	bit 7	
Bh	Excessive temperature	bit 6	reset by access **S**
Be	Excessive position error	bit 5	
Bw	Encoder wrap around	bit 4	
Bi	Index report available	bit 3	reset by access **I**
Bl	Historical negative limit	bit 2	reset by access **S**
Br	Historical positive limit	bit 1	reset by access **S**
Bt	Trajectory in progress	bit 0	

If **S** is reported, accessed or assigned, the historical bits, **Bl** and **Br**, are reset after the requested operation is completed. **S** may be monitored or periodically tested to check for unexpected conditions. If you are going to test **S** for various flag values, read **S** into a variable to avoid losing historical data and states. Since **S** reflects system states it is read only; **S=expression** is invalid; it will be ignored but it will cause a syntax error and set the extended system flag **Bs**.

Related Command:

RPW

RS

RW

EXAMPLE:

```
O=10000                 'set up move
P=0
A=222
V=33333
MP
G                       'go
WHILE Bt
      GOSUB5            'monitor for status change
LOOP
PRINT(#13," FINAL REPORT",#13)
GOSUB5                  'final report
END
C5
ss=S                    'READ VALUE ONCE
                        'to record historical latches
                        'before reset !
PRINT(#13," STATUS BYTE VALUE ", ss)
IF ss&32
      PRINT(#13," POSITION ERROR !!!")
ENDIF
IF ss&16
      PRINT(#13," WRAP AROUND !!!")
ENDIF
IF ss&1
      PRINT(#13," TRAJECTORY IN PROGRESS")
ENDIF
RETURN
```

Related Command:

ADDR

APPLICATION: Serial communication control

DESCRIPTION: Set motor address

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: Expression and conditional testing via **ADDR**

UNITS: Number

RANGE OF VALUES: **1** to **120**

TYPICAL VALUES: **1** to **4**

DEFAULT VALUE: **0**= global address

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The **SADDR{value}** command is used to set the unit address of a SmartMotor™, where "value" is an integer between 0 and 100. Separate addresses allow multiple SmartMotors to share a common communication channel and still differentiate themselves.

The **SADDR** command is typically one of the first commands in a downloaded program. In an RS-485 network, where all communications go over the same two parallel wires, the **SADDR** command must be in the program, whereas in an RS-232 network, where communications travel from one motor to the next, addressing can be accomplished from a host, or master motor.

The address can be from 0 to 100. If it is zero, the motor will have no unique address. Address 0 is the global address; it is used to talk to all motors on a network at once.

EXAMPLE:

```
SADDR1          'Set address to 1
```

When given a non-zero address, a SmartMotor begins to listen to commands after it receives its own unique address or the global address byte from the network. There is no need to repeat the address byte with subsequent commands intended for the same motor. The particular SmartMotor will continue to listen to commands until it receives a different address byte, after which commands are ignored. The echo function of the SmartMotor is not affected by the addressed state. That is, if told to echo, a SmartMotor will continue to echo, regardless of whether it is listening to commands.

Continued n next page:

Related Command:

ADDR

EXAMPLE:

```
'Example Auto Addressing for 4 SmartMotors™ via SADDR command
'on an RS-232 Daisy chain
'This program code would be run at the same time
'in all motors on the chain at power-up.

ECHO                                    ' Enable ECHO mode
a=1                                     ' User variable "a" to set
address.
WAIT=2000                         ' Wait about 1/2 second to allow
power-up to each motor
PRINT(#128,"a=a+1 ",#13)          'Print downstream to each motor
WAIT=2000                         ' Wait about 1/2 second for each motor
to ECHO
                                        ' through the same string to the
next motor

'Note: At this point, each motor will have run the exact same code
'causing successive motors downstream to receive the same command
string
'from the number of motors upstream.

SWITCH a                                ' Check he value of "a"
      CASE 1
            SADDR1                      ' Set Address to 1
            GOSUB10
      BREAK
      CASE 2
            SADDR2                      ' Set Address to 2
            GOSUB20
      BREAK

      CASE 3
            SADDR3                      ' Set Address to 3
            GOSUB30
      BREAK

      CASE 4
            SADDR4                      ' Set Address to 4
            GOSUB40
      BREAK
ENDS

END

C10 'MOTOR 1 CODE

RETURN
C20 'MOTOR 2 CODE

RETURN
C30 'MOTOR 3 CODE

RETURN
C40 'MOTOR 4 CODE

RETURN
```

SILENT

Silence Primary Port Outgoing Communications

Related Command:

TALK

TALK1

SILENT1

APPLICATION: Serial communication control

DESCRIPTION: Motor prevented from sending channel 0 responses to commands

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: **TALK0** state

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

These commands are almost always sent from a host, rather than existing within a program.

The **SILENT** command causes the SmartMotor™ to suppress all internally originating serial communication messages intended for the channel 0 primary port. It does not prevent the SmartMotor from sending messages in response to incoming serial report commands from the host, and it does not interfere with **ECHO**ing received serial communication over channel 0.

This command is most commonly used when sending a new program to an individual SmartMotor mounted in a networked system. In order to guarantee that the program arrives as sent, it is required that all other motors in the array be silent during download.

The **TALK** command negates the effect of **SILENT** and restores the motor's primary port to it's default state of operation.

SILENT1

Silence Secondary Port Outgoing Communications

Related Command:

TALK

TALK1

SILENT

APPLICATION: Serial communication control

DESCRIPTION: Motor prevented from sending channel 1 responses to commands

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: **TALK1** state

FIRMWARE VERSIONS: 4.0 and later

DETAILED DESCRIPTION:

The **SILENT1** command causes the SmartMotor™ to suppress all internally originating serial communication messages intended for the channel 1 secondary port. It does not prevent the SmartMotor from sending messages in response to incoming serial report commands from the host..

This command is most commonly used when sending a new program to an individual SmartMotor mounted in a networked system. In order to guarantee that the program arrives as sent, it is required that all other motors in the array be silent during download.

The **TALK1** command negates the effect of **SILENT1** and restores the motor's secondary port to it's default state of operation.

SIZE=expression

Set Number of CAM Table Data Points

Related Command:

BASE

G

MC

APPLICATION:	Mode CAM control
DESCRIPTION:	Number a data entries for CAM Mode
EXECUTION:	Buffered pending **MC** and **G** commands
CONDITIONAL TO:	N/A
LIMITATIONS:	**SIZE < BASE**
REPORT COMMAND:	None
READ/WRITE:	Write only
LANGUAGE ACCESS:	Assignment only
UNITS:	Encoder counts
RANGE OF VALUES:	0 to 32767
TYPICAL VALUES:	0 to 100
DEFAULT VALUE:	0
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

The SmartMotor™ performs a practical cam application by partitioning the required cam trajectory definition into a number of linearly interpolated segments. The variable **SIZE** stores the number of segments.

The segments are required to partition the **BASE** into a set of equally spaced intervals. For example; if **BASE=1000** and **SIZE=50**, each segment will then be 20 counts wide (**BASE/SIZE**).

The cam motion is then defined by providing the required SmartMotor positions corresponding to **CTR=0**, **20**, **40**, **60**, . . . **940**, **960**, **980** and **1000**. If the motion is truly periodic the required position at **CTR=0** will be identical to the required position at **CTR=1000**. The set of required positions are to be entered into the **aw[ ]** array, beginning at **aw[0]** and ending with **aw[SIZE]**. It is simplest to define the cam using position at **CTR=0** to be encoder position 0 by issuing **MF0** and **O=0** commands.

Related Command:

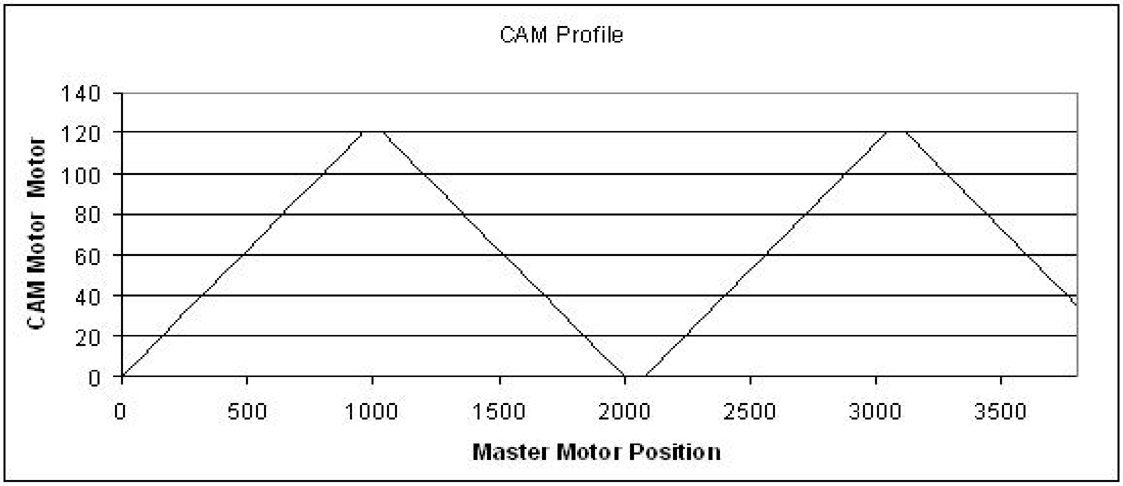
BASE

G

MC

EXAMPLE:

A "saw tooth" cam with periodic motion every 2000 external encoder counts and the motion interpolation divided into 25 (equal) segments.




```
BASE=2000   'Cam period
SIZE=25     'data segments (number of data points in table)
'CTR data interval = BASE/SIZE = 2000/25 = 80
'CAM motor will be at Data position every 80
'Master encoder counts:
'CTR=0, CTR=80, CTR=160,.... CTR=1840, CTR=1920, CTR=2000
'Now assigning data values beginning with aw[0]:
aw[0] 0 10 20 30 40 50 60 70 80 90 100.
aw[20] 110 120 120 110 100 90 80 70 60.
aw[19] 50 40 30 20 10 0.
MF4    'reset external encoder to zero
O=0    'reset internal encoder position
MC     'buffer CAM Mode
G      'start following the external encoder using cam data
```

The motor will now begin following the External (Master) encoder via the defined CAM profile above.

Related Command:

SLEEP1

WAKE

WAKE1

APPLICATION: Serial communication control

DESCRIPTION: Motor prevented from executing channel 0 commands

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: Illegal with a user program

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: **WAKE** state

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

*The **SLEEP** and **WAKE** commands are only sent from a host, never part of a SmartMotor™ program.*

The **SLEEP** command is used to put a SmartMotor™ into Sleep Mode with respect to channel 0 serial commands. While in Sleep Mode, a SmartMotor will continue to echo (if in **ECHO** mode) all characters received over the network, but will ignore all commands other than a **WAKE** command. A sleeping SmartMotor will also ignore a **G**-line input "go" request, but will be responsive to other input's dedicated functions.

The most common use of the **SLEEP** command is to keep daisy-chained SmartMotors from responding to commands in a program which is being downloaded to another SmartMotor™ in the same chain.

If a program is running when a SmartMotor receives the **SLEEP** command, the program will continue to run. Messages originating from within the running program of a sleeping SmartMotor will be transmitted unless the motor is also in **SILENT** mode.

SLEEP may be issued from the terminal or within a user program. **SLEEP** mode is terminated by the **WAKE** command.

Related Command:

SLEEP

WAKE

WAKE1

APPLICATION:	Serial communication control
DESCRIPTION:	Motor prevented from executing channel 1 commands
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Illegal with a user program
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	**WAKE1** state
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

The **SLEEP1** command is used to put a SmartMotor™ into Sleep Mode with respect to channel 1 serial commands. When in Sleep Mode, a SmartMotor will continue to echo (if in **ECHO1** mode) all characters received over the network, but will ignore all commands other than a **WAKE1** command. A sleeping SmartMotor will also ignore a **G**-line input "go" request, but will be responsive to other input's dedicated functions.

The most common use of the **SLEEP1** command is to keep SmartMotors in a daisy-chain from responding to commands imbedded in a program which is being downloaded to another SmartMotor in the same chain.

If a program is running when a SmartMotor™ receives the **SLEEP1** command, the program will continue to run. Messages originating from within the running program of a sleeping SmartMotor will be transmitted unless the motor is also in **SILENT1** mode.

SLEEP1 may be issued from the terminal or within a user program. **SLEEP1** mode is terminated by the **WAKE1** command.

Related Command:

END

GOSUB

RUN

RUN?

*Are there any **WHILE** statements in version 4.00?*

APPLICATION: Program execution control

DESCRIPTION: Reset user program subroutine return stack

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: Prior to version 4.00 a total of 6 **WHILE** and **GOSUB** statements are permitted at one time.

Version 4.00 supports up to 6 **GOSUB** statements at one time.

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Return **STACK** empty

FIRMWARE VERSIONS: **4.00 and higher**

DETAILED DESCRIPTION:

STACK empties the queue of pending (**GOSUB**) **RETURN** addresses.

In order to execute the **RETURN** program statement, the processor needs to be able to recall the program address point to which it should return. These addresses are stored within a region called a "stack".

A maximum of six address locations can be stored within the stack. This means that if a seventh **GOSUB** is called prior to any intervening **RETURN** statements, the stack will overflow and the program execution may fail. The stack region is managed using a pointer to the presently effective return address storage location. The **STACK** command directly resets this pointer to its initial condition. So the **STACK** command clears all **RETURN** addresses in the stack queue.

Note: Since Issuing **STACK** will cause any **RETURN** command to follow to be ignored , proper program flow via **GOTO** commands or otherwise should be used to prevent a memory mapping error. Care should be taken when the **STACK** command is used.

Since **GOSUB** command may be issued serially to the Smartmotor, it may be possible to overflow the stack regardless of the downloaded program code. The **STACK** can be issued via serial communications as well to permit the program execution to continue without concern for "how did we get here?". However, it is not recommended since full knowledge of what lin of code the motor may be running at the time wuuld not be known.

Related Command:

END

GOSUB

RUN

RUN?

EXAMPLE:

```
C0
GOTO1

C7
 PRINT(#13, "NO PROGRAM CRASH")
RETURN

END
GOSUB1
C1 GOSUB2
C2 GOSUB3
C3 GOSUB4
C4 GOSUB5
C5 GOSUB6          'sixth GOSUB without return
C6
 STACK             'reset internal stack
 GOSUB7            'allowing a seventh GOSUB
PRINT(#13,"RETURN FROM GOSUB7 OK")
END
```

The example above is not a good way to write code. It is just a means to explain where the **STACK** command would be used to prevent program crashes.
Often times, the **STACK** command is used after an error or motor protection fault is detected. Then immediately after the **STACK** command, either **RUN, END** or **GOTO**(location near top of program) is issued to recover.

Related Command:

BREAK

CASE number

DEFAULT

ENDS

APPLICATION:	Program execution control
DESCRIPTION:	Multiple choice branch for program execution
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	Can only be executed from within user program
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

The **SWITCH** command allows program flow control based on specific integer values of an expression or specific parameter or variable..

At execution runtime the program interpreter evaluates the **SWITCH expression** value and then tests the **CASE** numbers for a equal value in the order written in the program. If the expression value does not equal the **CASE** number, the next **CASE** statement is evaluated. If the expression value does equal the **CASE** number, program execution continues with the command immediately after. The execution time is similar to the equivalent **IF** expression control block. This means placing the most likely **CASE** values at the top of the **CASE** list will yield the faster average program execution time. The **DEFAULT** entry point is used if no **CASE** number is equals the expression value; it is executed last. If no **CASE** number equals the value of the **SWITCH expression** and there is no **DEFAULT** case, program execution passes through the **SWITCH** to the **ENDS** without performing any commands.

If a **BREAK** is encountered, program execution branches to the instruction or label following the **ENDS** of the **SWITCH** control block. **BREAK** can be used to isolate **CASE**s. Without **BREAK**, the **CASE** number syntax is transparent and program execution continues at the next instruction. That is, you will run into the next **CASE** number code sequence.

Each **SWITCH** control block must have at least one **CASE** number defined plus one, and only one, **ENDS** statement. **SWITCH** is not a valid terminal command, it is only valid within a user program.

Related Command:

BREAK

CASE number

DEFAULT

ENDS

Consider the following code fragment:

```
    SWITCH v
        CASE 1
            PRINT(" v = 1   ",#13)
            BREAK
        CASE 2
            PRINT(" v = 2   ",#13)
            BREAK
        CASE -23
            PRINT(" v = -23 ",#13)
            BREAK
        DEFAULT
            PRINT("v IS NOT 1,2 OR -23",#13)
        BREAK
    ENDS
```

The first line, **SWITCH v**, lets the SmartMotor™ know that it is checking the value of the variable **v**. Each following **CASE** begins the section of code that tells the SmartMotor what to do if **v** is equal to that "case".

EXAMPLE:

```
a=-3                                'test value
WHILE a<4
    PRINT(#13,"a=",a," ")
    SWITCH a                        'test expression
        CASE 3
            PRINT("MAX VALUE",#13)
        BREAK
        CASE -1             'negative test values are valid
        CASE -2             'note no BREAK here
        CASE -3
            PRINT("NEGATIVE")
        BREAK               'note use of BREAK

        CASE 0              'zero test value is valid
            PRINT("ZERO")       'note order is random

        DEFAULT             'the default case
            PRINT("NO MATCH VALUE")
        BREAK
    ENDS                    'need not be numerical
        a=a+1
    LOOP
    END
```

The output is

a=-3 NEGATIVE

a=-2 NEGATIVE

a=-1 NEGATIVE

a=0 ZERO

a=1 NO MATCH VALUE

a=2 NO MATCH VALUE

a=3 MAX VALUE

T=expression

Set Open Loop Commanded Torque Value

Related Command:

MT

RT

APPLICATION:	Motion mode control
DESCRIPTION:	Torque value for **MODE TORQUE**
EXECUTION:	Immediate
CONDITIONAL TO:	**MT** issued
LIMITATIONS:	N/A
REPORT COMMAND:	**RT**
READ/WRITE:	Read write
LANGUAGE ACCESS:	Assignment, expressions and conditional testing
UNITS:	Fraction of available torque
RANGE OF VALUES:	**-1023** to **1023**
TYPICAL VALUES:	**-1000** and **1000**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

Command **MT** enables torque mode. In this mode, the motor is commanded to develop a specific power level, set by **T=expression**. **T** is in units of Tenths of Percent of the full capacity of the subject motor and takes values between **-1023** and **1023**. **T=-1023** results in full torque in the negative direction. The encoder still tracks position and can still be read with the **@P** variable, but the **PID** loop is off and the motor is not servoing or running a trajectory.

MT sets the **PWM** signal to the drive to a fixed percentage, which means that the amplifier tries to deliver a fixed amount of power to the motor coils. For any given torque and no applied load, there will be a velocity at which the back **EMF** of the motor will cause the acceleration to stop and the velocity to hold more or less constant. Under the no load or static load conditions, the **T** command will control velocity. As the load torque increases, the velocity decreases.

Note: This means that **MT** does not regulate torque. Instead, it delivers a fixed amount of power to the motor coils. As motor power is the product of torque and RPM, velocity decreases as the delivered torque increases and vice versa.

In all firmware 4.76, MT will immediately turn on the servo and reset any position error. The servo-off flag **Bo** is set to **0**, the trajectory flag **Bt** is reset to **0**, and the position error flag **Be** is reset to **0**. The motion is not restricted by the present **E** value. Issuing **E=0** would have no effect upon the present motion. The drive stage is still subject to the currently defined activity of the limit switches.

In all firmware >=476, any prior faults must be cleared prior to accepting the MT command.

Continued on next page:

Related Command:

MT

RT

Amplifier mode **MD50** DOES EFFECT the value of **T**. To change from mode **MD50** to mode **MT**, issue the sequence **OFF T=value MT**.

EXAMPLE:

```
      UAI                 'Set I/O A as Input
      T=0                 'Initialize T=0
      MT                  'Enter Mode Torque
      C1                  'Label 1, Loop Forever
      a=UAA-512           'Read User defined I/O pin
                          '10 bit analog reading range
                          'is 0 to 1023 from 0 to 5VDC
                          '[ 2.5 V = 0 Torque ]
      T=2*a
' Result: -1023 to +1023 values from 0 to 5VDC

      GOTO1               'GOTO LABEL 1

      END
```

The above example will track an incoming analog signal from 0 to 5 Volts (**UAA=0** to **1023**) and assign it to the **T** torque value of -1023 to 1023.

TALK

Enable Outgoing Messages on Primary Port

Related Command:

SILENT

SILENT1

TALK1

APPLICATION:	Serial communication control
DESCRIPTION:	Normal channel 0 communications mode
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	**TALK** state
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

These commands are almost always sent from a host, rather than existing within a program.

TALK restores the motor’s ability to print messages to the serial communication channel 0 if that ability was previously suppressed with the **SILENT** command. This command is most commonly used following the download a user program to a SmartMotor™ within a daisy chain. It could also be used to "un-silence" a debug routine.

TALK may be issued from the terminal or within a user program.

Related Command:

SILENT

SILENT1

TALK

APPLICATION:	Serial communication control
DESCRIPTION:	Normal channel 1 communications mode
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	**TALK1** state
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

TALK1 restores the motor's ability to print messages to the serial communication channel 1 if that ability was previously suppressed with the **SILENT1** command. This command is most commonly used following the download a user program to a SmartMotor™ within a daisy chain. It could also be used to "un-silence" a debug routine.

TALK1 may be issued from the terminal or within a user program.

TEMP

Read Motor Temperature

Related Command:

BH

RBh

TH

THD

APPLICATION: Temperature control

DESCRIPTION: Read motor temperature

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: Read Only

LANGUAGE ACCESS: N/A

UNITS: Degrees Centigrade

RANGE OF VALUES: **-128** to **127**

TYPICAL VALUES: **20** to **60**

DEFAULT VALUE: Room temperature

FIRMWARE VERSIONS: 4.11 and higher

DETAILED DESCRIPTION:

The present temperature of the motor can be determined by assigning **TEMP** to a user variable or issuing **PRINT(TEMP)**. The units are degrees Centigrade. **EXAMPLE:**

```
t=TEMP
Rt            'response 30

PRINT(TEMP) 'response 31 - the motor is warming up
```

Motors with version 4.11 and higher permit the user to set the overheat temperature trip point with the command **TH=expression**, and to set the time (**THD=expression**) for which the overheat condition must exist before the servo is shut off. A motor in the overheat condition will not turn on the servo even if commanded to do so.

If the motor were operating in **Torque Mode** at **TEMP>TH** for 4 seconds, the motor would shut off. It would not restart until both the condition **TH-TEMP>5** were true and then **MT** command reissued.

```
a=-5
WHILE a<=10
      TH=TEMP+a
      WAIT=4000
      G
      WAIT=4000
      IF Bt
            BREAK
      ENDIF
      a=a+1
LOOP
PRINT("MOTOR RESTARTED WHEN TH-TEMP=",a)
END
```

Restart announced at **TH - TEMP = 6**.

Related Command:

Bh

RBh

TEMP

THD

APPLICATION: Temperature control

DESCRIPTION: Set maximum allowable temperature limit

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: Read write

LANGUAGE ACCESS: N/A

UNITS: Degrees Centigrade

RANGE OF VALUES: **0** to **70**

TYPICAL VALUES: **20** to **60**

DEFAULT VALUE: **70 or 85 (model number dependant)**

FIRMWARE VERSIONS: 4.11 and higher

DETAILED DESCRIPTION:

TH=expression sets the maximum allowable temperature at which the SmartMotor™ is permitted to continually servo. The amount of time that the SmartMotor can still servo at or above this temperature is set by the **THD** function. If the temperature stays at or above the **TH** value for longer than **THD** servo samples, the amplifier will turn off, **Bh** will be set to **1**, the motor off bit **Bo** set to 1 and the trajectory bit cleared to 0. If issued, **RMODE** will return "**O**." The SmartMotor will reject any command to start motion until the temperature has fallen 5º Celsius.

There is no direct report command for **TH**, but *variable*=**TH** and **PRINT(TH)** are both valid.

EXAMPLE: (demonstrates relationship between **TEMP**, **TH**, and **Bh**)

```
GOSUB10            'report TEMP, TH, and Bh

a=5
WHILE a>-5         'vary TH about the present TEMP
      TH=TEMP-a
      WAIT=2000
      GOSUB10      'observe Bh flag change from o to 1
      a=a-1        'as TH is reduced to TEMP value and
                         less
LOOP

END

C10
      PRINT(#13,"Read the temperature   ",TEMP)
      PRINT(#13,"Read TH overheat value ",TH)
      PRINT(#13,"Read Bh overheat flag  ",Bh)
RETURN
```

Related Command:

Bh

RBh

TEMP

TH

APPLICATION: Temperature control

DESCRIPTION: Set overheat delay time

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: Write only

LANGUAGE ACCESS: N/A

UNITS: **PID** samples

RANGE OF VALUES: **0** to **65536**

TYPICAL VALUES: **12000**

DEFAULT VALUE: **12000** samples, approximately 3 seconds

FIRMWARE VERSIONS: 4.11 and higher

DETAILED DESCRIPTION:

The THD command permits the user to set to set the time for which the overheat condition may exist before the servo is shut off. **THD=16000** means that the SmartMotor ™ will allows an overheat condition for 16000 servo samples or approximately 4 seconds before shutting down. The maximum value for **THD** is 20**000**, in 4.4x series firmware and 64**000** in all others. One Servo Sample is ~ 250useconds.

If an overheat condition exists for more than **THD** samples, the amplifier will turn off, **Bh** will be set to 1, the motor off bit **Bo** set to 1 and the trajectory bit cleared to 0. If issued, **RMODE** will return "**O**." The SmartMotor will reject any command to start motion until the temperature has fallen 5º Celsius.

EXAMPLE: (test to measure approximate shut down time - not very accurate but illustrates **TH**, **THD**, and **TEMP**)

```
PRINT(#13,"Default value of TH = ",TH)
PRINT(#13,"Motor Temperature = ",TEMP)
PRINT(#13,"START MOTION")
A=222
V=44444
MV
G
THD=32000    'THD default = 12000 PID samples or 3 seconds
TH=TEMP-5    'Force an over heat condition
             'Units are degrees Centigrade
a=CLK
WHILE Bh==0 LOOP
WHILE Bt LOOP
b=CLK
PRINT(#13,"Servo OFF after ",b-a," PID samples")
END
```

Pause Program Execution During Active Trajectory

Related Command:

WAIT=exp

APPLICATION: Program execution control

DESCRIPTION: Suspend command execution while in trajectory

EXECUTION: Immediate

CONDITIONAL TO: **Bt** state

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The **TWAIT** command will pause program execution until the Busy Trajectory status bit clears. Normally, program execution and trajectory generation are completely independent. Regardless of what the motion is doing, the processor executed ode form the top down. If there were three consecutive motion commands they would all execute sequentially. Before the motor could even start to move, last motion command would dominate. Using the **TWAIT** command, however, allows the move commands to occur and complete end to end. An alternative to **TWAIT** is **WHILE Bt . . . LOOP**.

Both **TWAIT** and the **WHILE Bt** construction terminate when the trajectory ends, regardless of the cause. Depending on the application, you may wish to perform error checking to ensure that the move was properly completed.

EXAMPLE:

```
C100        'Motion Subroutine
      MP          'Mode Position
      A=100       'Set acceleration
      V=10000     'Set velocity
      P=2000      'Set first position
      G           'Start Motion
      TWAIT       'wait till trajectory is done
      P=-4000     'Set next position
      G           'Start Motion
      WHILE Bt    'While moving (similar to TWAIT)
            IF UA==0
                  GOSUB200
            ENDIF
      LOOP  'wait till trajectory is done
RETURN   'Return to GOSUB
```

UA=expression

Set I/O Port A Out t Logi c State

Related Command:

UAA

UAI

UAO

APPLICATION: I/O control

DESCRIPTION: Set Pin A output latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: Write only

LANGUAGE ACCESS: Assignment only

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0** or **1**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

With this function you could actually check if your output is shorted.

User I/O line A can function as a TTL output. The pin defaults to be a general purpose TTL (0 - 5 volt) input. To use PIN A as an output, set the value of the pin A output latch **UA** to either **0** or **1**. Issue the command **UAO** if this has not already been issued.

I/O pin A will be a logic high voltage if **UA=1** and a logic low voltage if **UA=0**.

Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UAA** function.

EXAMPLE:

```
UAO         'set PIN A to function as a digital output
UA=0        'set PIN A to logic 0 (zero volts)
UA=1        'set PIN A to logic 1 (+5 volts)
```

Note: The I/O state can be set prior to assigning as an output.

```
UA=0        'Pre-set PIN A to logic 0 (zero volts)
UAO         'set PIN A as an output pre-initialized to zero
```

Related Command:

UA

UAI

UAO

APPLICATION:	I/O control
DESCRIPTION:	Read PIN A analog input
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	PRINT(UAA)
READ/WRITE:	Read only
LANGUAGE ACCESS:	Expression and conditional testing
UNITS:	Number
RANGE OF VALUES:	**0** to **1023**
TYPICAL VALUES:	**0** to **1023**
DEFAULT VALUE:	I/O dependent
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

User I/O line A can serve as a 10 bit analog to digital input. The A to D reference is 5VDC and the returned data is between 0 and 1023. A value of 0 corresponds to 0 volts and 1023 to 5 volts. **UAA** is read only, and can be accessed with the statement *variable***=UAA**, **PRINT(UAA,#13)** or **WHILE UAA>200 . . . LOOP**. The analog read occurs once at the time the **UAA** command is executed. Assigning the variable **a=UAA** will perform the analog read once and store it into the variable **a**.

All user I/O pins have in internal 5K pull-up resistor, as well as current limiting and other protection mechanisms. Any analog voltage source, then, should be rated to adequately drive a 5K ohm input impedance.

The analog to digital conversion is always available on its corresponding I/O pin. That is, regardless of whether the pin is being used as an input, output or other function, a 10 bit analog reading of I/O that pin is always available.

EXAMPLE:

```
PRINT(#13,"PRINT UAA = ",UAA)
b=UAA
PRINT(#13,"REPORT UAA = ")
Rb
```

```
RUAA  'Directly Report Port A Analog Value (>=4.76 firmware only)
```

UAI (as command)
Set I/O Port A to Input

Related Command:

UA

UAA

UAO

APPLICATION: I/O control

DESCRIPTION: Set Pin A to be an input

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

User I/O line A serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input, secondary encoder input A or the step input in Step and Direction Mode. While, user I/O line A defaults to being a general purpose TTL input, it can be explicitly set up as a digital input with the **UAI** command.

If I/O line A has been set to an output with the command **UAO**, it can be reset to be an input with the command **UAI**.

EXAMPLE:

```
UAI            'Initialize (U)ser defined I/O pin (A) as (I)nput
PRINT(#13,"PIN A Input ",UAI)
n=UAI          'Store state of I/O pin A
               'as digital input into variable name "n"
PRINT(#13,"REPORT PIN A Input ") Rn
END
```

UAI (as input value)

Read I/O Port A Logic State

Related Command:

UA

UAA

UAO

APPLICATION: I/O input

DESCRIPTION: Input at Pin A

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **PRINT(UAI)**

READ/WRITE: Read only

LANGUAGE ACCESS: Expression and conditional testing

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0** or **1**

DEFAULT VALUE: I/O dependent

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

User I/O line A serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input, secondary encoder input A or the step input in Step and Direction Mode. User I/O line A defaults to being a general purpose TTL input. It can be accessed with the statement **variable=UAI**, **PRINT(UAI,#13)** or **WHILE UAI ... LOOP**. The digital read occurs once at the time the **UAI** command is executed. Assigning the variable **a=UAI** will perform the digital read once and store it into the variable **a**.

If I/O line A has been set to an output with the command **UAO**, it can be reset to be an input with the command **UAI**.

EXAMPLE:

```
      UAI          'Initialize (U)ser defined I/O pin (A) as (I)nput
      PRINT(#13,"PIN A Input ",UAI)
      n=UAI        'Store state of I/O pin A
                   'as digital input into variable name "n"
      PRINT(#13,"REPORT PIN A Input ") Rn
      END
```

```
RUA    'Directly Report Port A logic State  (>=4.76 firmware only)
```

```
n=U&1 'Bitmask Port A to the variable n,   (>=4.76 firmware only)
Rn    'Report Result
```

UAO (as command)

Set I/O Port A to Output

Related Command:

UA

UAA

UAI

APPLICATION: I/O control

DESCRIPTION: Set Pin A to be an output

EXECUTION: Immediate

CONDITIONAL TO: **UA=0** or **UA=1**

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

User I/O line A can function as a TTL output. The pin defaults to be a general purpose TTL (0 - 5 volt) input. The command **UAO** specifies the I/O pin A as an output, while **UA=value** sets the voltage. I/O pin A will be a logic high voltage if **UA=1** and a logic low voltage if **UA=0**. Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UAA** function.

In order for the output voltage to reflect the state of **UA**, both **UAO** and **UA=value** have to be issued. Suppose the I/O pin is functioning as a digital input. If you want to output a logic low signal, the pin will not sink current until both **UAO** and **UA=0** have been issued. You only have to issue **UAO** once; the I/O pin stays configured as an output for some other configuration specification is issued.

EXAMPLE:

```
UAO                   'define PIN A output
UA=1                  'set output latch value
PRINT(UAO)            'recall the latch value.
                      'response is 1
UA=0                  'set output latch value
PRINT(UAO)            'recall the latch value
                      'response is 0
```

UB=expression

Set I/O Port B Output Logic State

Related

UBA

UBI

UBO

APPLICATION:	I/O control
DESCRIPTION:	Set Pin B output latch
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	Write only
LANGUAGE ACCESS:	Assignment only
UNITS:	Binary bit
RANGE OF VALUES:	**0** or **1**
TYPICAL VALUES:	**0** or **1**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

User I/O line B can function as a TTL output. The pin defaults to be a general purpose TTL (0 - 5 volt) input. To use PIN B as an output, set the value of the pin B output latch **UB** to either **0** or **1**. Issue the command **UBO** if this has not already been issued.

I/O pin A will be a logic high voltage if **UB=1** and a logic low voltage if **UB=0**.

Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UBA** function.

EXAMPLE:

```
UBO         'set PIN B to function as a digital output
UB=0        'set PIN B to logic 0 (zero volts)
UB=1        'set PIN B to logic 1 (+5 volts)
```

Note: The I/O state can be set prior to assigning as an output.

```
UB=0        'Pre-set PIN B to logic 0 (zero volts)
UBO         'set PIN B as an output pre-initialized to zero
```

Related Command:

UB

UBI

UBO

APPLICATION:	I/O input
DESCRIPTION:	Read Pin B analog input
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	**PRINT(UBA)**
READ/WRITE:	Read only
LANGUAGE ACCESS:	Expression and conditional testing
UNITS:	Number
RANGE OF VALUES:	**0** or **1023**
TYPICAL VALUES:	**0** or **1023**
DEFAULT VALUE:	I/O dependent
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

User I/O line B can serve as a 10 bit analog to digital input. The A to D reference is 5VDC and the returned data is between 0 and 1023. A value of 0 corresponds to 0 volts and 1023 to 5 volts. **UBA** is read only, and can be accessed with the statement *variable***=UBA**, **PRINT(UBA,#13)** or **WHILE UBA>200 . . . LOOP**. The analog read occurs once at the time the **UBA** command is executed. Assigning the variable **a=UBA** will perform the analog read once and store it into the variable **a**.

All user I/O pins have in internal 5K pull-up resistor, as well as current limiting and other protection mechanisms. Any analog voltage source, then, should be rated to adequately drive a 5K ohm input impedance.

The analog to digital conversion is always available on its corresponding I/O pin. That is, regardless of whether the pin is being used as an input, output or other function, a 10 bit analog reading of I/O that pin is always available.

EXAMPLE:

```
PRINT(#13,"PRINT UBA = ",UBA)
b=UBA
PRINT(#13,"REPORT UBA = ")
Rb
```

```
RUBA  'Directly Report Port B Analog Value (>=4.76 firmware only)
```

UBI (as command)

Set I/O Port B to Input

Related Command:

UB

UBA

UBO

APPLICATION: I/O control

DESCRIPTION: Set Pin B to be an input

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

User I/O line B serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input, secondary encoder input B or the direction input in Step and Direction Mode. While user I/O line B defaults to being a general purpose TTL input, it can be explicitly set up as a digital input with the **UBI** command.

If I/O line B has been set to an output with the command **UBO**, it can be reset to be an input with the command **UBI**.

EXAMPLE:

```
UBI          'Initialize (U)ser defined I/O pin (B) as (I)nput
PRINT(#13,"PIN B Input ",UBI)
n=UBI        'Store state of I/O pin B
             'as digital input into variable name "n"
PRINT(#13,"REPORT PIN B Input ") Rn
END
```

UBI (as input value)

Read I/O Port B Logic State

Related Command:

UB

UBA

UBO

APPLICATION: I/O input

DESCRIPTION: Input at Pin B

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **PRINT(UBI)**

READ/WRITE: Read only

LANGUAGE ACCESS: Expression and conditional testing

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0** or **1**

DEFAULT VALUE: I/O dependent

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

User I/O line B serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input, secondary encoder input B or the direction input in Step and Direction Mode. User I/O line B defaults to being a general purpose TTL input. It can be accessed with the statement **variable=UBI**, **PRINT(UBI,#13)** or **WHILE UBI . . . LOOP**. The digital read occurs once at the time the **UBI** command is executed. Assigning the variable **a=UBI** will perform the digital read once and store it into the variable a.

If I/O line B has been set to an output with the command **UBO**, it can be reset to be an input with the command **UBI**.

EXAMPLE:

```
UBI          'Initialize (U)ser defined I/O pin (B) as (I)nput
PRINT(#13,"PIN B Input ",UBI)
n=UBI        'Store state of I/O pin B
             'as digital input into variable name "n"
PRINT(#13,"REPORT PIN B Input ") Rn
END
```

```
RUB   'Directly Report Port B logic State  (>=4.76 firmware only)
```

```
n=U&2 'Bitmask Port B to the variable n,   (>=4.76 firmware only)
Rn    'Report Result
```

UBO (as command)

Set I/O Port B to Output

Related Command:

UB

UBA

UBI

APPLICATION: I/O control

DESCRIPTION: Set Pin B to be an output

EXECUTION: Immediate

CONDITIONAL TO: **UB=0** or **UB=1**

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

User I/O line B can function as a TTL output. The pin defaults to be a general purpose TTL (0 - 5 volt) input. The command **UBO** specifies the I/O pin B as an output, while **UB=value** sets the voltage. I/O pin B will be a logic high voltage if **UB=1** and a logic low voltage if **UB=0**. Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UBA** function.

In order for the output voltage to reflect the state of **UB**, both **UBO** and **UB=value** have to be issued. Suppose the I/O pin is functioning as a digital input. If you want to output a logic low signal, the pin will not sink current until both **UBO** and **UB=0** have been issued. You only have to issue **UBO** once; the I/O pin stays configured as an output for some other configuration specification is issued.

EXAMPLE:

```
UBO                 'define PIN B output
UB=1                'set output latch value
PRINT(UBO)          'recall the latch value.
                    'response is 1
UB=0                'set output latch value
PRINT(UBO)          'recall the latch value
                    'response is 0
```

Related Command:

UCA

UCI

UCO

APPLICATION: I/O control

DESCRIPTION: Set Pin C output latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: Write only

LANGUAGE ACCESS: Assignment only

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0** or **1**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line C can function as a TTL output. The pin defaults to be a general purpose TTL (0 - 5 volt) input. To use PIN C as an output, set the value of the pin C output latch **UC** to either **0** or **1**. Issue the command **UCO** if this has not already been issued.

I/O pin C will be a logic high voltage if **UC=1** and a logic low voltage if **UC=0**.

Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UCA** function.

EXAMPLE:

```
UCO         'set PIN C to function as a digital output
UC=0        'set PIN C to logic 0 (zero volts)
UC=1        'set PIN C to logic 1 (+5 volts)
```

Note: The I/O state can be set prior to assigning as an output.

```
UC=0        'Pre-set PIN C to logic 0 (zero volts)
UCO         'set PIN C as an output pre-initialized to zero
```

Read I/O Port C as Analog Input

Related Command:

UC

UCI

UCO

APPLICATION: I/O control

DESCRIPTION: Read Pin C analog input

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **PRINT(UCA)**

READ/WRITE: Read only

LANGUAGE ACCESS: Expression and conditional testing

UNITS: Number

RANGE OF VALUES: **0** or **1023**

TYPICAL VALUES: **0** or **1023**

DEFAULT VALUE: I/O dependent

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line C can serve as a 10 bit analog to digital input. The A to D reference is 5VDC and the returned data is between 0 and 1023. A value of 0 corresponds to 0 volts and 1023 to 5 volts. **UCA** is read only, and can be accessed with the statement *variable*=**UCA**, **PRINT(UCA,#13)** or **WHILE UCA>200 . . . LOOP**. The analog read occurs once at the time the **UCA** command is executed. Assigning the variable **a=UCA** will perform the analog read once and store it into the variable **a**.

All user I/O pins have in internal 5K pull-up resistor, as well as current limiting and other protection mechanisms. Any analog voltage source, then, should be rated to adequately drive a 5K ohm input impedance.

The analog to digital conversion is always available on its corresponding I/O pin. That is, regardless of whether the pin is being used as an input, output or other function, a 10 bit analog reading of I/O that pin is always available.

EXAMPLE:

```
PRINT(#13,"PRINT UCA = ",UCA)
b=UCA
PRINT(#13,"REPORT UCA = ")
Rb

RUCA  'Directly Report Port C Analog Value (>=4.76 firmware only)
```

Related Command:

UC

UCA

UCO

APPLICATION: I/O control

DESCRIPTION: Set Pin C to be an input

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line C serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input and the positive travel limit input. While user I/O line C defaults to being the positive limit input, it can be explicitly set up as a digital input with the **UCI** command.

If I/O line C has been set to an output with the command **UCO**, it can be reset to be an input with the command **UCI**.

EXAMPLE:

```
UCI           'Initialize (U)ser defined I/O pin (C) as (I)nput
PRINT(#13,"PIN C Input ",UCI)
n=UCI         'Store state of I/O pin C
              'as digital input into variable name "n"
PRINT(#13,"REPORT PIN C Input ") Rn
END
```

UCI (as input value)

Read I/O Port C to Input

Related Command:

UC

UCA

UCO

APPLICATION: I/O input

DESCRIPTION: Input at Pin C

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **PRINT(UCI)**

READ/WRITE: Read only

LANGUAGE ACCESS: Expression and conditional testing

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0** or **1**

DEFAULT VALUE: I/O dependent

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line C serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input, and Defaults to the positive travel limit input. It can be accessed with the statement **variable=UCI**, **PRINT(UCI,#13)** or **WHILE UCI . . . LOOP**. The digital read occurs once at the time the **UCI** command is executed. Assigning the variable **a=UCI** will perform the digital read once and store it into the variable **a**.

EXAMPLE:

```
      UCI          'Initialize (U)ser defined I/O pin (C) as (I)nput
      PRINT(#13,"PIN C Input ",UCI)
      n=UCI        'Store state of I/O pin C
                   'as digital input into variable name "n"
      PRINT(#13,"REPORT PIN C Input ") Rn
      END
```

```
RUC    'Directly Report Port C logic State   (>=4.76 firmware only)
```

```
n=U&4 'Bitmask Port C to the variable n,    (>=4.76 firmware only)
Rn     'Report Result
```

UCO (as command)

Set I/O Port C to Output

Related Command:

UC

UCA

UCI

APPLICATION: I/O control

DESCRIPTION: Set Pin C to be an output

EXECUTION: Immediate

CONDITIONAL TO: **UC=0** or **UC=1**

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Although its default function is to be the right limit input, user I/O line C can function as a TTL output. The command **UCO** specifies the I/O pin C as an output, while **UC=value** sets the voltage. I/O pin C will be a logic high voltage if **UC=1** and a logic low voltage if **UC=0**. Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UCA** function.

In order for the output voltage to reflect the state of **UC**, both **UCO** and **UC=value** have to be issued. Suppose the I/O pin is functioning as a digital input. If you want to output a logic low signal, the pin will not sink current until both **UCO** and **UC=0** have been issued. You only have to issue **UCO** once; the I/O pin stays configured as an output for some other configuration specification is issued.

EXAMPLE:

```
UCO                 'define PIN C output
UC=1                'set output latch value
PRINT(UCO)          'recall the latch value.
                    'response is 1
UC=0                'set output latch value
PRINT(UCO)          'recall the latch value
                    'response is 0
```

Set I/O Port C as Positive Over Travel Limit

Related Command:

LIMD

LIMH

LIML

LIMN

UC

UCA

UCI

APPLICATION: I/O control

DESCRIPTION: Set PIN C to be right / positive limit input

EXECUTION: Immediate

CONDITIONAL TO: **UC=0** or **UC=1**

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Limit switch

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line C can be a TTL (0 to 5V) input, TTL output, 10 bit input, or act as the positive limit input, which is the default state. **UCP** explicitly defines I/O pin C to be the positive limit, while commands **UCI** and **UCO** make it into a TTL input or output, respectively, disabling the limit behavior.

EXAMPLE:

```
UCI           'use PIN C as a general purpose input
              'suppress limit behavior
a=UCI         'read the input value as digital input
Ra            'report input value
UCP           'restore default positive limit behavior to PIN C
```

UD=expression

Set I/O Port D Output Logic State

Related Command:

UDA

UDI

UDO

APPLICATION: I/O control

DESCRIPTION: Set Pin D output latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: Write only

LANGUAGE ACCESS: Assignment only

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0** or **1**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line D can function as a TTL output. The pin defaults to be a general purpose TTL (0 - 5 volt) input. To use PIN D as an output, set the value of the pin D output latch **UD** to either **0** or **1**. Issue the command **UDO** if this has not already been issued.

I/O pin D will be a logic high voltage if **UD=1** and a logic low voltage if **UD=0**.

Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UDA** function.

EXAMPLE:

```
UDO          'set PIN D to function as a digital output
UD=0         'set PIN D to logic 0 (zero volts)
UD=1         'set PIN D to logic 1 (+5 volts)
```

Note: The I/O state can be set prior to assigning as an output.

```
UD=0         'Pre-set PIN D to logic 0 (zero volts)
UDO          'set PIN D as an output pre-initialized to zero
```

UDA

Read I/O Port D as Analog Input

Related Command:

UD

UDI

UDO

APPLICATION: I/O control

DESCRIPTION: Read Pin D analog input

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **PRINT(UDA)**

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Number

RANGE OF VALUES: **0** or **1023**

TYPICAL VALUES: **0** or **1023**

DEFAULT VALUE: I/O dependent

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line D can serve as a 10 bit analog to digital input. The A to D reference is 5VDC and the returned data is between 0 and 1023. A value of 0 corresponds to 0 volts and 1023 to 5 volts. **UDA** is read only, and can be accessed with the statement *variable*=**UDA**, **PRINT(UDA,#13)** or **WHILE UDA>200 . . . LOOP**. The analog read occurs once at the time the **UDA** command is executed. Assigning the variable **a=UDA** will perform the analog read once and store it into the variable **a**.

All user I/O pins have in internal 5K pull-up resistor, as well as current limiting and other protection mechanisms. Any analog voltage source, then, should be rated to adequately drive a 5K ohm input impedance.

The analog to digital conversion is always available on its corresponding I/O pin. That is, regardless of whether the pin is being used as an input, output or other function, a 10 bit analog reading of I/O that pin is always available.

EXAMPLE:

```
PRINT(#13,"PRINT UDA = ",UDA)
b=UDA
PRINT(#13,"REPORT UDA = ")
Rb
```

```
RUDA  'Directly Report Port D Analog Value (>=4.76 firmware only)
```

UDI (as command)

Set I/O Port D to Input

Related Command:

UD

UDA

UDM

UDO

APPLICATION: I/O control

DESCRIPTION: Set Pin D to be an input

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line D serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input and the negative travel limit input. While user I/O line D defaults to being the negative limit input, it can be explicitly set up as a digital input with the **UDI** command.

If I/O line D has been set to an output with the command **UDO**, it can be reset to be an input with the command **UDI**.

EXAMPLE:

```
UDI          'Initialize (U)ser defined I/O pin (D) as (I)nput
PRINT(#13,"PIN D Input ",UDI)
n=UDI        'Store state of I/O pin D
             'as digital input into variable name "n"
PRINT(#13,"REPORT PIN D Input ") Rn
END
```

UDI (as input value)

Read I/O Port D to Input

Related Command:

UD

UDA

UDM

UDO

APPLICATION: I/O input

DESCRIPTION: Input at Pin D

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **PRINT(UDI) [RUDI >-v4.76]**

READ/WRITE: Read only

LANGUAGE ACCESS: Expression and conditional testing

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0** or **1**

DEFAULT VALUE: I/O dependent

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line D serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input, and Defaults to being the negative travel limit input. It can be accessed with the statement **variable=UDI**, **PRINT(UDI,#13)** or **WHILE UDI . . . LOOP**. The digital read occurs once at the time the **UDI** command is executed. Assigning the variable **a=UDI** will perform the digital read once and store it into the variable **a**.

EXAMPLE:

```
UDI          'Initialize (U)ser defined I/O pin (D) as (I)nput
PRINT(#13,"PIN D Input ",UDI)
n=UDI        'Store state of I/O pin D
             'as digital input into variable name "n"
PRINT(#13,"REPORT PIN D Input ") Rn
END
```

```
RUD    'Directly Report Port D logic State  (>=4.76 firmware only)
```

```
n=U&8 'Bitmask Port D to the variable n,   (>=4.76 firmware only)
Rn     'Report Result
```

UDM

Set I/O Port D as Negative Over Travel Limit

Related Command:

LIMH

LIML

LIMN

UD

UDA

UDI

APPLICATION: I/O control

DESCRIPTION: Set Pin D to be left/negative limit input

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT STATE: Limit switch

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line D can be a TTL (0 to 5V) input, TTL output, 10 bit input, or act as the negative limit input, which is the default state. UDM explicitly defines I/O pin D to be the negative limit, while commands **UDI** and **UDO** make it into a TTL input or output, respectively, disabling the limit behavior.

EXAMPLE:

```
UDI         'Initialize PIN D as a general purpose input
            'suppress limit behavior
a=UDI       'read the input value as a digital value
Ra          'report input value
UDM         'restore default negative limit behavior to PIN D
```

UDO (as command)

Set I/O Port D to Output

Related Command:

UD

UDA

UDI

APPLICATION: I/O control

DESCRIPTION: Set Pin D to be an output

EXECUTION: Immediate

CONDITIONAL TO: **UD=0** or **UD=1**

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Although its default function is to be the left limit input, user I/O line D can function as a TTL output. The command **UDO** specifies the I/O pin D as an output, while **UD=value** sets the voltage. I/O pin D will be a logic high voltage if **UD=1** and a logic low voltage if **UD=0**. Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UDA** function.

In order for the output voltage to reflect the state of **UD**, both **UDO** and **UD=value** have to be issued. Suppose the I/O pin is functioning as a digital input. If you want to output a logic low signal, the pin will not sink current until both **UDO** and **UD=0** have been issued. You only have to issue **UDO** once; the I/O pin stays configured as an output for some other configuration specification is issued.

EXAMPLE:

```
UDO                 'define PIN D output
UD=1                'set output latch value
PRINT(UDO)          'recall the latch value.
                    'response is 1
UD=0                'set output latch value
PRINT(UDO)          'recall the latch value
                    'response is 0
```

UE=expression

Set I/O Port E Output Logic State

Related Command:

UEA

UEI

UEO

APPLICATION: I/O control

DESCRIPTION: Set Pin E output latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: Write only

LANGUAGE ACCESS: Assignment only

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0** or **1**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line E can function as a TTL output. The pin defaults to be a general purpose TTL (0 - 5 volt) input. To use PIN E as an output, set the value of the pin E output latch **UE** to either **0** or **1**. Issue the command **UEO** if this has not already been issued.

I/O pin E will be a logic high voltage if **UE=1** and a logic low voltage if **UE=0**.

Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UEA** function.

EXAMPLE:

```
UEO          'set PIN E to function as a digital output
UE=0         'set PIN E to logic 0 (zero volts)
UE=1         'set PIN E to logic 1 (+5 volts)
```

Note: The I/O state can be set prior to assigning as an output.

```
UE=0         'Pre-set PIN E to logic 0 (zero volts)
UEO          'set PIN E as an output pre-initialized to zero
```

UEA
Read I/O Port E as Analog Input

Related Command:

UE

UEI

UEO

APPLICATION: I/O control

DESCRIPTION: Read Pin E analog input

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **PRINT(UEA)**

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Number

RANGE OF VALUES: **0** or **1023**

TYPICAL VALUES: **0** or **1023**

DEFAULT VALUE: I/O dependent

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line E can serve as a 10 bit analog to digital input. The A to D reference is 5VDC and the returned data is between 0 and 1023. A value of 0 corresponds to 0 volts and 1023 to 5 volts. **UEA** is read only, and can be accessed with the statement *variable***=UCE**, **PRINT(UEA,#13)** or **WHILE UEA>200 . . . LOOP**. The analog read occurs once at the time the **UEA** command is executed. Assigning the variable **a=UEA** will perform the analog read once and store it into the variable **a**.

All user I/O pins have in internal 5K pull-up resistor, as well as current limiting and other protection mechanisms. Any analog voltage source, then, should be rated to adequately drive a 5K ohm input impedance.

The analog to digital conversion is always available on its corresponding I/O pin. That is, regardless of whether the pin is being used as an input, output or other function, a 10 bit analog reading of I/O that pin is always available.

EXAMPLE:

```
PRINT(#13,"PRINT UEA = ",UEA)
b=UEA
PRINT(#13,"REPORT UEA = ")
Rb
```

```
RUEA  'Directly Report Port E Analog Value (>=4.76 firmware only)
```

UEI (as command)

Set I/O Port E to Input

Related Command:

UE

UEA

UEO

APPLICATION: I/O control

DESCRIPTION: Set Pin E to be an input

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line E serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input, the AniLink data line and the RS485 A signal. While user I/O line E defaults to being the AniLink data line, it can be explicitly set up as a digital input with the **UEI** command.

If I/O line E has been set to an output with the command **UEO**, it can be reset to be an input with the command **UEI**.

EXAMPLE:

```
      UEI          'Initialize (U)ser defined I/O pin (E) as (I)nput
      PRINT(#13,"PIN E Input ",UEI)
      n=UEI        'Store state of I/O pin E
                   'as digital input into variable name "n"
      PRINT(#13,"REPORT PIN E Input ") Rn
      END
```

```
RUE    'Directly Report Port E logic State  (>=4.76 firmware only)


n=U&16 'Bitmask Port E to the variable n,   (>=4.76 firmware only)
Rn     'Report Result
```

UEI (as input value)

Set I/O Port E to Input

Related Command::

UE

UEA

UEO

APPLICATION:	I/O input
DESCRIPTION:	Input at Pin E
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	**PRINT(UEI)**
READ/WRITE:	Read only
LANGUAGE ACCESS:	Expression and conditional testing
UNITS:	Binary bit
RANGE OF VALUES:	**0** or **1**
TYPICAL VALUES:	**0** or **1**
DEFAULT VALUE:	I/O dependent
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

User I/O line E serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input, the AniLink data line and the RS485 A signal. While user I/O line E defaults to being the AniLink data line, it can be explicitly set up as a digital input with the **UEI** command.

If I/O line E has been set to an output with the command **UEO**, it can be reset to be an input with the command **UEI**.

EXAMPLE:

```
      UEI          'Initialize (U)ser defined I/O pin (E) as (I)nput
      PRINT(#13,"PIN E Input ",UEI)
      n=UEI        'Store state of I/O pin E
                   'as digital input into variable name "n"
      PRINT(#13,"REPORT PIN E Input ") Rn
      END
```

```
RUE   'Directly Report Port E logic State  (>=4.76 firmware only)


n=U&16 'Bitmask Port E to the variable n,   (>=4.76 firmware only)
Rn     'Report Result
```

UEO (as command)

Set I/O Port E to Input

Related Command:

UE

UEA

UEI

APPLICATION: I/O control

DESCRIPTION: Set Pin E to be an output

EXECUTION: Immediate

CONDITIONAL TO: **UE=0** or **UE=1**

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Although its default function is to be the AniLink data line, user I/O line E can function as a TTL output. The command **UEO** specifies the I/O pin E as an output, while **UE=value** sets the voltage. I/O pin E will be a logic high voltage if **UE=1** and a logic low voltage if **UE=0**. Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UEA** function.

In order for the output voltage to reflect the state of **UE**, both **UEO** and **UE=value** have to be issued. Suppose the I/O pin is functioning as a digital input. If you want to output a logic low signal, the pin will not sink current until both **UEO** and **UE=0** have been issued. You only have to issue **UEO** once; the I/O pin stays configured as an output for some other configuration specification is issued.

EXAMPLE: (set PIN E as output and recall output latch value)

```
UEO                  'define PIN E output
UE=1                 'set output latch value
PRINT(UEO)           'recall the latch value.
                     'response is 1
UE=0                 'set output latch value
PRINT(UEO)           'recall the latch value
                     'response is 0
```

UF=expression

Set I/O Port F Output Logic State

Related Command:

UFA

UFI

UFO

APPLICATION: I/O control

DESCRIPTION: Set Pin F output latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: Write only

LANGUAGE ACCESS: Assignment only

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0** or **1**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line F can function as a TTL output. The pin defaults to be a general purpose TTL (0 - 5 volt) input. To use PIN F as an output, set the value of the pin F output latch **UF** to either **0** or **1**. Issue the command **UFO** if this has not already been issued.

I/O pin F will be a logic high voltage if **UF=1** and a logic low voltage if **UF=0**.

Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UFA** function.

FXAMPLE:

```
UFO          'set PIN F to function as a digital output
UF=0         'set PIN F to logic 0 (zero volts)
UF=1         'set PIN F to logic 1 (+5 volts)
```

Note: The I/O state can be set prior to assigning as an output.

```
UF=0         'Pre-set PIN F to logic 0 (zero volts)
UFO          'set PIN F as an output pre-initialized to zero
```

Related Command::

UF

UFI

UFO

APPLICATION: I/O control

DESCRIPTION: Read Pin F analog input

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **PRINT(UFA)**

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Number

RANGE OF VALUES: **0** or **1023**

TYPICAL VALUES: **0** or **1023**

DEFAULT VALUE: I/O dependent

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line F can serve as a 10 bit analog to digital input. The A to D reference is 5VDC and the returned data is between 0 and 1023. A value of 0 corresponds to 0 volts and 1023 to 5 volts. **UFA** is read only, and can be accessed with the statement *variable*=**UFA**, **PRINT(UFA,#13)** or **WHILE UFA>200 . . . LOOP**. The analog read occurs once at the time the **UFA** command is executed. Assigning the variable **a=UFA** will perform the analog read once and store it into the variable **a**.

All user I/O pins have in internal 5K pull-up resistor, as well as current limiting and other protection mechanisms. Any analog voltage source, then, should be rated to adequately drive a 5K ohm input impedance.

The analog to digital conversion is always available on its corresponding I/O pin. That is, regardless of whether the pin is being used as an input, output or other function, a 10 bit analog reading of I/O that pin is always available.

EXAMPLE:

```
PRINT(#13,"PRINT UCA = ",UFA)
b=UFA
PRINT(#13,"REPORT UFA = ")
Rb
```

```
RUFA  'Directly Report Port F Analog Value (>=4.76 firmware only)
```

UFI (as command)

Set I/O Port F to Input

Related Command:

UF

UFA

UFO

APPLICATION: I/O control

DESCRIPTION: Set Pin F to be an input

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line F serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input, the AniLink clock line and the RS485 B signal. While user I/O line F defaults to being the AniLink clock line, it can be explicitly set up as a digital input with the **UFI** command.

If I/O line F has been set to an output with the command **UFO**, it can be reset to be an input with the command **UFI**.

EXAMPLE:

```
UFI          'Initialize (U)ser defined I/O pin (F) as (I)nput
PRINT(#13,"PIN F Input ",UFI)
n=UFI        'Store state of I/O pin F
             'as digital input into variable name "n"
PRINT(#13,"REPORT PIN F Input ") Rn
END
```

```
RUF   'Directly Report Port F logic State  (>=4.76 firmware only)
```

```
n=U&32 'Bitmask Port F to the variable n,   (>=4.76 firmware only)
Rn     'Report Result
```

UFI (as input value)

Read I/O Port F Logic State

Related Command::

UF

UFA

UFO

APPLICATION: I/O input

DESCRIPTION: Input at Pin F

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **PRINT(UFI)**

READ/WRITE: Read only

LANGUAGE ACCESS: Expression and conditional testing

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0** or **1**

DEFAULT VALUE: I/O dependent

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line F serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input, the AniLink clock line and the RS485 B signal. While user I/O line F defaults to being the AniLink clock line, it can be explicitly set up as a digital input with the **UFI** command.

If I/O line F has been set to an output with the command **UFO**, it can be reset to be an input with the command **UFI**.

EXAMPLE:

```
      UFI          'Initialize (U)ser defined I/O pin (F) as (I)nput
      PRINT(#13,"PIN E Input ",UFI)
      n=UFI        'Store state of I/O pin F
                   'as digital input into variable name "n"
      PRINT(#13,"REPORT PIN F Input ") Rn
      END
```

```
RUF   'Directly Report Port F logic State  (>=4.76 firmware only)
```

```
n=U&32 'Bitmask Port F to the variable n,   (>=4.76 firmware only)
Rn     'Report Result
```

UFO (as command)

Set I/O Port F to Output

Related Command:

UF

UFA

UFI

APPLICATION: I/O control

DESCRIPTION: Set Pin F to be an output

EXECUTION: Immediate

CONDITIONAL TO: **UF=0** or **UF=1**

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Although its default function is to be the AniLink clock line, user I/O line F can function as a TTL output. The command UFO specifies the I/O pin F as an output, while **UF=value** sets the voltage. I/O pin F will be a logic high voltage if **UF=1** and a logic low voltage if **UF=0**. Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UFA** function.

In order for the output voltage to reflect the state of **UF**, both **UFO** and **UF=value** have to be issued. Suppose the I/O pin is functioning as a digital input. If you want to output a logic low signal, the pin will not sink current until both **UFO** and **UF=0** have been issued. You only have to issue **UFO** once; the I/O pin stays configured as an output for some other configuration specification is issued.

EXAMPLE: (set PIN F as output and recall output latch value)

```
UFO                 'define PIN F output
UF=1                'set output latch value
PRINT(UFO)          'recall the latch value.
                    'response is 1
UF=0                'set output latch value
PRINT(UFO)          'recall the latch value
                    'response is 0
```

Enable/Re-Enable Port G Sync Functionality

Related Command:

UGA

UGI

UGO

RS4

APPLICATION:	I/O control
DESCRIPTION:	Set Pin G to Act as "G" command when grounded
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	Assignment only
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	**N/A**
DEFAULT VALUE:	**N/A**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

User I/O line G can function as the "GO" or G command when grounded. It does so by default. If at any time UGI or UGO commands are used, this functionality is disabled. To Re-enable the "sync-function" just issue UG by itself.

The reason it is called the "sync function" is because it allows multiple motors to trigger Go commands via hardware at the exact same time thereby synchronizing them.

UG=expression

Set I/O Port G Output Logic State

Related Command:

UGA

UGI

UGO

RS4

APPLICATION: I/O control

DESCRIPTION: Set Pin G output latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: Write only

LANGUAGE ACCESS: Assignment only

UNITS: Binary bit

RANGE OF VALUES: **0** or **1**

TYPICAL VALUES: **0** or **1**

DEFAULT VALUE: **0**

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

User I/O line G can function as a TTL output. The pin defaults to be a general purpose TTL (0 - 5 volt) input. To use PIN G as an output, set the value of the pin G output latch **UG** to either **0** or **1**. Issue the command **UGO** if this has not already been issued.

I/O pin G will be a logic high voltage if **UG=1** and a logic low voltage if **UG=0**.

Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of that I/O pin is always available through the **UGA** function.

GXAMPLE:

```
UGO         'set PIN G to function as a digital output
UG=0        'set PIN G to logic 0 (zero volts)
UG=1        'set PIN G to logic 1 (+5 volts)
```

Note: The I/O state can be set prior to assigning as an output.

```
UG=0        'Pre-set PIN G to logic 0 (zero volts)
UGO         'set PIN G as an output pre-initialized to zero
```

UGA (as input value)

Read I/O Port G As Analog Input

Related Command:

UG

UGI

UGO

APPLICATION: I/O control

DESCRIPTION: Read Pin G analog input

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **PRINT(UGA)**

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Number

RANGE OF VALUES: **0** or **1023**

TYPICAL VALUES: **0** or **1023**

DEFAULT VALUE: I/O dependent

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line G can serve as a 10 bit analog to digital input. The A to D reference is 5VDC and the returned data is between 0 and 1023. A value of 0 corresponds to 0 volts and 1023 to 5 volts. **UGA** is read only, and can be accessed with the statement *variable***=UGA**, **PRINT(UGA,#13)** or **WHILE UGA>200 . . . LOOP**. The analog read occurs once at the time the **UGA** command is executed. Assigning the variable **a=UGA** will perform the analog read once and store it into the variable **a**.

All user I/O pins have in internal 5K pull-up resistor, as well as current limiting and other protection mechanisms. Any analog voltage source, then, should be rated to adequately drive a 5K ohm input impedance.

The analog to digital conversion is always available on its corresponding I/O pin. That is, regardless of whether the pin is being used as an input, output or other function, a 10 bit analog reading of I/O that pin is always available.

EXAMPLE:

```
PRINT(#13,"PRINT UGA = ",UGA)
b=UGA
PRINT(#13,"REPORT UGA = ")
Rb
```

```
RUGA  'Directly Report Port G Analog Value (>=4.76 firmware only)
```

UGI (as input value)

Read I/O Port G Logic Level State

Related Command:

UG

UGI

UGO

APPLICATION: I/O control

DESCRIPTION: Read Pin G Logicinput

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **PRINT(UGI)**

READ/WRITE: Read only

LANGUAGE ACCESS: Expressions and conditional testing

UNITS: Number

RANGE OF VALUES: **0** or **1023**

TYPICAL VALUES: **0** or **1023**

DEFAULT VALUE: I/O dependent

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

User I/O line G serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input, the hardware "go" line, and the primary port RS485 control line. While user I/O line **G** defaults to being the active low hardware "go," it can be explicitly set up as a digital input with the **UGI** command.

If I/O line G has been set to an output with the command **UGO**, it can be reset to be an input with the command **UGI**.

EXAMPLE:

```
UGI          'Initialize (U)ser defined I/O pin (G) as (I)nput
PRINT(#13,"PIN E Input ",UGI)
n=UGI        'Store state of I/O pin G
             'as digital input into variable name "n"
PRINT(#13,"REPORT PIN G Input ") Rn
END
```

```
RUG    'Directly Report Port G logic State  (>=4.76 firmware only)
```

```
n=U&64 'Bitmask Port G to the variable n,   (>=4.76 firmware only)
Rn     'Report Result
```

UGI (as command)
Set I/O Port G to Input

Related Command:

UG

UGA

UGO

RS4

APPLICATION:	I/O control
DESCRIPTION:	Set PIN G to be an input
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT STATE:	Input
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

User I/O line G serves many functions. It can be a TTL (0 to 5V) input, TTL output, 10 bit analog input, the hardware "go" line, and the primary port RS485 control line. While user I/O line **G** defaults to being the active low hardware "go," it can be explicitly set up as a digital input with the **UGI** command.

If I/O line G has been set to an output with the command **UGO**, it can be reset to be an input with the command **UGI**.

EXAMPLE:

```
      UGI          'Initialize (U)ser defined I/O pin (G) as (I)nput
      PRINT(#13,"PIN G Input ",UGI)
      n=UGI        'Store state of I/O pin G
                   'as digital input into variable name "n"
      PRINT(#13,"REPORT PIN G Input ") Rn
      END


RUG   'Directly Report Port G logic State  (>=4.76 firmware only)


n=U&64 'Bitmask Port G to the variable n,   (>=4.76 firmware only)
Rn     'Report Result
```

UGO (as command)

Set I/O Port G to Output

Related Command:

UG

UGA

UGI

RS4

APPLICATION: I/O control

DESCRIPTION: Set Pin G to be an output

EXECUTION: Immediate

CONDITIONAL TO: **UG=0** or **UG=1**

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

DEFAULT VALUE: Input

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

Although its default function is the hardware "go" line, user I/O line G can function as a TTL output. The command **UGO** specifies the I/O pin G as an output, while **UG=value** sets the voltage. I/O pin G will be a logic high voltage if **UG=1** and a logic low voltage if **UG=0**. Regardless of whether the I/O pin is being used as an input or output, a 10 bit analog reading of the I/O pin is always available through the **UGA** function.

In order for the output voltage to reflect the state of **UG**, both **UGO** and **UG=value** have to be issued. Suppose the I/O pin is functioning as a digital input. If you want to output a logic low signal, the pin will not sink current until both **UGO** and **UG=0** have been issued. Just issue **UGO** once, the I/O pin stays configured until another configuration specification is issued.

When you open channel 0 as an RS485 port dedicates I/O G to the RS485 control function, which is required for use with Animatics RS232 to RS485 converters like the RS485 and RS485-ISO. When using one of these adapters, you must ensure that the I/O G pin is configured as a TTL output with the **UGO** command before the channel is opened.

EXAMPLE:

```
UGO                 'define PIN G output
UG=1                'set output latch value
PRINT(UGO)          'recall the latch value.
                    'response is 1
UG=0                'set output latch value
PRINT(UGO)          'recall the latch value
                    'response is 0
```

Related Command::

UPLOAD

APPLICATION:	User program verification
DESCRIPTION:	Upload user EEPROM through serial communications
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	ASCII Characters
RANGE OF VALUES:	Alpha numeric
TYPICAL VALUES:	Alpha numeric
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

WARNING

Do not use the ***UP*** *command within a user program.*

It will terminate the program.

The **UP** command will cause the SmartMotor™ compiled user program runtime code to be sent out the primary serial port. In contrast, the **UPLOAD** command returns the user program in readable text. The output from the **UP** command will include a header containing binary information and special codes, created by the compiler to make the program run faster, interspersed with the program text.

UP immediately terminates any running user program. The program counter is lost. **UP** does not terminate the present motion mode or trajectory, change motion parameters such as **E**, **A**, **V**, or **KP**, or alter the present value of the user variables.

The comments in your original source code do not appear when you **UP** or **UPLOAD** a program. Comments are removed by the compiler, which is normal for any compiled computer program.

When uploading a program from a SmartMotor in a daisy chain, prevent the other SmartMotors in the chain from issuing unexpected characters by using the **SILENCE** and **SLEEP** commands. After the upload is complete, you can re-enable normal communications with **WAKE** and **TALK**.

Related Command::

UP

APPLICATION:	User program verification
DESCRIPTION:	Upload user EEPROM through serial communications
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	**UPLOAD** terminates user program execution
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	ASCII Characters
RANGE OF VALUES:	Alpha numeric
TYPICAL VALUES:	Alpha numeric
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

WARNING

*Do not use the **UPLOAD** command within a user program.*

It will terminate the program.

The **UPLOAD** command will upload only the text portion of the SmartMotor's™ program as it appeared in your original source file. In contrast, the **UP** command will upload the text along with all of the binary information created by the compiler that allows the program to run faster.

UPLOAD immediately terminates any running user program. The program counter is lost. **UPLOAD** does not terminate the present motion mode or trajectory, or change motion parameters such as **E**, **A**, **V**, **KP,** etc., or alter the present value of the users variables.

When communicating over a terminal use the **UPLOAD** command to verify the program is the expected one. The comments in your original source code do not appear when you **UP** or **UPLOAD** a program. The comments were removed by the compiler, as is usual for any compiled computer program.

When uploading a program from a SmartMotor in a daisy chain, prevent the other SmartMotors in the chain from issuing unexpected characters by using the **SILENCE** and **SLEEP** commands. After the upload is complete, you can re-enable normal communications with WAKE and TALK.

EXAMPLE: (try the following program, down load it and then **RUN**)

```
PRINT(" PERFORM UPLOAD CMD")
UPLOAD
PRINT(" ANY MORE ?")
END
```

Output is "PERFORM UPLOAD CMD"

Related Command:

@P
@PE
@V
A
D
E
G
MP
MV
V

APPLICATION:	Trajectory control
DESCRIPTION:	Maximum velocity
EXECUTION:	Buffered
CONDITIONAL TO:	**MP**, **MV**
LIMITATIONS:	N/A
REPORT COMMAND:	**PRINT(V)**
READ/WRITE:	Read write
LANGUAGE ACCESS:	Assignment, expressions, and conditional testing
UNITS:	Scaled encoder counts
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-23200000** to **3200000**
DEFAULT VALUE:	**0**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

Use the **V=expression** to set the slew rate used by the velocity and position mode moves. In the SmartMotor™, a point to point move is determined by **P=expression**, the target position, **V=expression**, the target travelling velocity, and **A=expression**, the acceleration at which to reach the target velocity. In a velocity mode move, you only need **V=expression**, the target travelling velocity, and **A=expression**, the acceleration at which to reach the target velocity. **V** is always positive in position mode but can be positive or negative in velocity mode.

The value of **V** defaults to zero so it must be given a value before any motion can take place. The new value does not take effect until the next **G** command is executed.

```
MP                      'Set Position Mode
P=10000                 'Set Position
V=10000                 'Set Velocity
A=1000                  'Set Acceleration
G                       'Start Motion
TWAIT                   'pause program execution during move
P=0                     'Set new position
G                       'Start Motion again
```

Velocity is held to 32 bits, 16 bits integer and 16 bits fractional. The units are counts per sample period, shifted by the 16 bits (65,536).

$$32{,}212 = \frac{(2{,}000\text{counts/revolution})(65{,}536)}{(4{,}069\text{samples/second})}$$

VLD(*variable, number*)

Data EEPROM READ/WRITE COMMAND

Related Command:

Bk

EPTR

RBk

VST

APPLICATION:	User data recovery
DESCRIPTION:	Sequentially load user variables from data EPROM
EXECUTION:	Immediate
CONDITIONAL TO:	**EPTR=** variable
LIMITATIONS:	**EPTR** set from 0 to 32000
REPORT COMMAND:	N/A
READ/WRITE:	Sequential read
LANGUAGE ACCESS:	N/A
UNITS:	1 byte, 2 byte, or 4 byte reads
RANGE OF VALUES:	-2147483648 to 2147483647
TYPICAL VALUES:	-2147483648 to 2147483647
DEFAULT VALUE:	User stored values
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

VST( ) or **VLD( )** commands are used to *store* and *load* data from internal nonvolatile RAM, (EEPROM). To read or write into this memory space a memory address location must first be specified with the **EPTR=expression** command, where expression takes a value between **0** and **32000**, and then use the **VST( )** or **VLD( )** commands to store or retrieve data.

To Read in a series of values and assign these values to a sequence of user variables use the **VLD(***variable, number***)** command.
The first parameter (*variable*) specifies the name of the first user variable of a sequence of variables that you wish to load.
The second parameter (*number*) specifies the number of variables in the sequence of variables that you wish to store.
The command interpreter will automatically note the size of variable you define, either 1, 2, or 4 bytes long.

When using the data EEPROM, it is important to note that the only the data values are stored or loaded. The association of these values to any variable is not retained. The only way to retrieve this data is by keeping track of the **EPTR** value.

If the data memory access is out of range, the scan error flag **Bs** will be set.

EXAMPLES:

Storing and retrieving a single 32 bit standard variable:

```
a=123456778    'assign a value to the variable "a"
EPTR=100       'Set EPROM pointer to 100
VST(a,1)       'Store into EPROM (EPTR incremental to 104 automatically)
EPTR=100       'Set Eprom to 100 again
VLD(b,1)       'Load from location 100 into the variable "b"
Rb             'Report result will be:     123456789
```

Related Command:

Bk

EPTR

RBk

VST

Storing and retrieving a single 16 bit standard variable:

```
aw[0]=32000   'assign a value to the 16 bit "array word"(0)
EPTR=100      'Set Eprom pointer to 100
VST(aw[0],1)  'Store into EPROM (EPTR incremental to 102 automatically)
EPTR=100      'Set Eprom to 100 again
VLD(x,1)      'Load from location 100 into the variable "x"
Rx            'Report result will be:     32000
```

Storing and retrieving a single 8 bit standard variable:

```
ab[0]=126     'assign a value to the 8 bit "array byte"(0)
EPTR=100      'Set Eprom pointer to 100
VST(aw[0],1)  'Store into EPROM  EPTR incremental to 101 automatically)
EPTR=100      'Set Eprom to 100 again
VLD(x,1)      'Load from location 100 into the variable "x"
Rx            'Report result will be:     126
```

Storing and retrieving a 5 consecutive 32 bit standard variables:

```
a 10 11 12 13 14.         'assign values to the variables "a" thru "f"
EPTR=100      'Set Eprom pointer to 100
VST(a,5)      'EPTR will increment to 100+(4*5)=120
              '(4 bytes  x 5 stored)
EPTR=100      'Set Eprom to 100 again
VLD(v,5)      'Load from location 100 into the variable "b"
Rv                'will report 10
Rw                'will report 11
Rx                'will report 12
Ry                'will report 13
Rz                'will report 14
```

Storing 7 16-bit numbers into EEPROM:

```
      i=10   'Using the variable "i" as index to an array variable
      j=7    'Using the variable "j" as the number of sequential
             'variables you wish to store
```

I've left the strikethroughs intact. I assume there'll be something to replace them or they'll go away eventually

. . . Ernie

Example 16-bit array data Data :

```
      aw[i] 1111 2222 3333 4444 -1111 -2222 -3333.
      EPTR=3200          'Set EPROM memory pointer location to 3200
      VST(aw[i],j)       'Store "j" or 7 sequential variables
                         'beginning with aw[i]
                         'into EPROM starting at address 3200.
```

Note: The EEPROM value will automatically increment for each value stored.
EPTR value after above execution will be set to
3200+(7 variable * 2 bytes each) or 3214

Retrieving Same data into other variables for later use:

```
      EPTR=3200
      i=10   'Using the variable "i" as index to an array variable
      j=7    'Using the variable "j" as the number of sequential
             'variables you wish to store
      VLD(aw[r],s)
      WHILE t<5
            PRINT(#13,aw[t+r]," ")
            t=t+1
      LOOP
      END                'output is 111 222 333 444 -1111
```

VST(*variable, number*)

DATA-EEPROM READ/WRITE COMMAND

Related Command:

Bk

EPTR

RBk

VST

APPLICATION:	User data storage
DESCRIPTION:	Sequentially store user variables to data EPROM
EXECUTION:	Immediate
CONDITIONAL TO:	**EPTR=** variable
LIMITATIONS:	**EPTR** set from **0** to **7999**
REPORT COMMAND:	N/A
READ/WRITE:	Sequential write
LANGUAGE ACCESS:	N/A
UNITS:	1 byte, 2 byte, or 4 byte reads
RANGE OF VALUES:	**-2147483648** to **2147483647**
TYPICAL VALUES:	**-2147483648** to **2147483647**
DEFAULT VALUE:	User determined values
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

VST( ) command is used to *store* data into internal nonvolatile RAM, (EEPROM). To write into this memory space a memory address location must first be specified with the **EPTR=expression** command, where expression takes a value between **0** and **32000**, use the **VST(***variable, number***)** command. The first parameter (*variable*) specifies the name of the first user variable of a sequence of variables that you wish to write from. The second parameter (*number*) specifies the number of variables in the sequence of variables that you wish to store.

The command interpreter will automatically note the size of variable you define, either 1, 2, or 4 bytes long.

When using the data EEPROM, it is important to note that the only the data values are stored. The association of these values to any variable is not retained. The only way to retrieve this data is by keeping track of the **EPTR** value.

As each byte is written to the EEPROM, is immediately verified by reading the EEPROM device. If the byte read does not match the byte write the system bit **Bk** will be set to **1**. If the data memory access is out of range, the scan error flag **Bs** will be set.

EXAMPLES:

Storing and retrieving a single 32 bit standard variable:

```
a=123456778    'assign a value to the variable "a"
EPTR=100       'Set EPROM pointer to 100
VST(a,1)       'Store into EPROM (EPTR incremental to 104 automatically)
EPTR=100       'Set Eprom to 100 again
VLD(b,1)       'Load from location 100 into the variable "b"
Rb             'Report result will be:     123456789
```

Related Command:

Bk

EPTR

RBk

VST

Storing and retrieving a single 16 bit standard variable:

```
aw[0]=32000   'assign a value to the 16 bit "array word"(0)
EPTR=100      'Set Eprom pointer to 100
VST(aw[0],1)  'Store into EPROM (EPTR incremental to 102 automatically)
EPTR=100      'Set Eprom to 100 again
VLD(x,1)      'Load from location 100 into the variable "x"
Rx            'Report result will be:     32000
```

Storing and retrieving a single 8 bit standard variable:

```
ab[0]=126     'assign a value to the 8 bit "array byte"(0)
EPTR=100      'Set Eprom pointer to 100
VST(aw[0],1)  'Store into EPROM  EPTR incremental to 101 automatically)
EPTR=100      'Set Eprom to 100 again
VLD(x,1)      'Load from location 100 into the variable "x"
Rx            'Report result will be:     126
```

Storing and retrieving a 5 consecutive 32 bit standard variables:

```
a 10 11 12 13 14.         'assign values to the variables "a" thru "f"
EPTR=100      'Set Eprom pointer to 100
VST(a,5)      'EPTR will increment to 100+(4*5)=120
              '(4 bytes  x 5 stored)
EPTR=100      'Set Eprom to 100 again
VLD(v,5)      'Load from location 100 into the variable "b"
Rv                'will report 10
Rw                'will report 11
Rx                'will report 12
Ry                'will report 13
Rz                'will report 14
```

Storing 7 16-bit numbers into EEPROM:

```
      i=10   'Using the variable "i" as index to an array variable
      j=7    'Using the variable "j" as the number of sequential
             'variables you wish to store
```

Example 16-bit array data Data :

```
      aw[i] 1111 2222 3333 4444 -1111 -2222 -3333.
      EPTR=3200          'Set EPROM memory pointer location to 3200
      VST(aw[i],j)       'Store "j" or 7 sequential variables
                         'beginning with aw[i]
                         'into EPROM starting at address 3200.
```

Note: The EEPROM value will automatically increment for each value stored. EPTR value after above execution will be set to 3200+(7 variable * 2 bytes each) or 3214

Retrieving Same data into other variables for later use:

```
      EPTR=3200
      i=10   'Using the variable "i" as index to an array variable
      j=7    'Using the variable "j" as the number of sequential
             'variables you wish to store
      VLD(aw[r],s)
      WHILE t<5
            PRINT(#13,aw[t+r]," ")
            t=t+1
      LOOP
      END                'output is 111 222 333 444 -1111
```

WAIT=expression

Pause Program Flow for pre-determined time

Related Command:

TWAIT

CLK

PID#

APPLICATION: Program execution control

DESCRIPTION: Suspends command execution for defined number of **PID** samples

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: N/A

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: **PID** samples

RANGE OF VALUES: **0** to **2147483647**

TYPICAL VALUES: **0** to **4000**

DEFAULT VALUE: N/A

FIRMWARE VERSIONS: ALL

DETAILED DESCRIPTION:

The **WAIT=expression** will pause program execution for a specified amount of time. Time is measured in **PID** sample periods of which there are 4,069 per second by default. Some firmware versions may have a different of **PID** rate - please refer to the **RSP** command for details on how to query your SmartMotor™ for its **PID** sample period. The number of **PID** sample periods per second can be changed with the **PID#** commands for motors with version 4.00 or later firmware.

EXAMPLE: (pause program execution for a given period)

```
     w=32552              'use to set Wait time

     PID1                 'Default PID updates every servo sample
     WAIT=w               'Wait time     = 8    seconds

     PID2                 'PID updates  every 2 servo samples
     WAIT=w               'Wait time     = 4    seconds

     PID4                 'PID updates  every 4 servo samples
     WAIT=w               'Wait time     = 2    seconds

     PID8                 'PID updates  every 8 servo samples
     WAIT=w               'Wait time     = 1    second

'
     PID1                 'Return to Default PID
     WAIT=w               'Wait time   = 8 seconds
```

WAKE

Enable Open Communications on Primary Port

Related Command:

SLEEP

SLEEP1

WAKE1

APPLICATION:	Serial communication control
DESCRIPTION:	Motor to execute all communications channel 0 commands
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	**WAKE** state
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

WAKE clears the **SLEEP** condition of a SmartMotor™. A SmartMotor that has been put to **SLEEP** rejects all commands received through the primary port but **WAKE**.

*The **SLEEP** and **WAKE** commands are only sent from a host, never part of a SmartMotor™ program.*

WAKE is intended to be used from the host terminal while programs are being downloaded to other motors, but is is perfectly valid from within a user program.

WAKE1

Enable Open Communications on Secondary Port

Related Command:

SLEEP

SLEEP1

WAKE1

APPLICATION:	Serial communication control
DESCRIPTION:	Motor to execute all communications channel 1 commands
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	**WAKE1** state
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

WAKE1 clears the **SLEEP1** condition of a SmartMotor™. A SmartMotor that has been put to **SLEEP1** rejects all commands received through the channel 1 serial port but **WAKE1**.

WAKE1 is intended to be used from the host terminal while programs are being downloaded to other motors, but is is perfectly valid from within a user program.

WHILE expression

Conditional Program Loop Flow Control

Related Command:

BREAK

LOOP

IF

SWITCH

APPLICATION:	Program execution control
DESCRIPTION:	Defines block of code repeatable while expression is true
EXECUTION:	Immediate
CONDITIONAL TO:	Value of expression
LIMITATIONS:	6 Deep WHILE loop nesting <v4.0 firmware No limit >=v4.0 firmware
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	expression values **-2147483648** to **2147483647**
TYPICAL VALUES:	expression values **-2147483648** to **2147483647**
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The WHILE loop creates a program loop that repeatedly executes as long as a certain condition is true or non zero.

EXAMPLE:

WHILE {*expression is true}*

execute program command here

LOOP

The "*expression*" is evaluated the first time **WHILE** is encountered, and each time program execution is sent back to the **WHILE** by its corresponding **LOOP** statement. If the "*expression*" value is zero or false, program execution re-directs to the code just below the **LOOP** command. Any valid standard Animatics expression can be used. In particular, W**HILE 1 . . . LOOP** is a standard loop forever control block.

Each **WHILE** expression control block must be terminated with a corresponding **LOOP** exit statement. **WHILE** control blocks may be nested.

If **BREAK** is encountered while executing a **WHILE** control block, program execution unconditionally takes up after the **LOOP** statement.

WHILE is not a valid terminal command, it is only valid within a user program.

SEE EXAMPLES ON NEXT PAGE

Related Command:

BREAK

LOOP

IF

SWITCH

EXAMPLE:

```
    WHILE Bt     'While trajectory still in progress
                 'More efficient than Bt==1
          UB=1   'Set output high
          UB=0   'Set output low
    LOOP         'Loop back to While
```

EXAMPLE:

```
    a=0
    WHILE a<7
          b=a<3        'this is valid syntax !

          IF b
                PRINT("T ")  'true !
          ELSE
                PRINT("F ")  'false !
          ENDIF
          a=a+1                    'increment loop index
    LOOP
    END
'output is "T T T F F F F "
```

EXAMPLE OF NESTED WHILE LOOPS:

```
D=20000             'Set Relative Move Distance
A=100               'Set Acceleration
V=1000000           'Set Velocity
MP                  'Set to Position Mode

 WHILE 1            'While Forever

    WHILE UAI==1 LOOP
    'wait for Port A to be grounded

    G               'Start Relative Move

    WHILE Bt     'While Moving
          IF UBI==0    'If Port B is grounded
                X      'Stop motion
          ENDIF
    LOOP

    WHILE UAI==0 LOOP
    'wait for Port A to reset.
    IF UCI==0          'If Port C was grounded
          BREAK        'exit the WHILE 1 LOOP
    ENDIF

  LOOP
  PRINT("Port C was grounded"),#13)

END
```

Related Command:

G

S

APPLICATION:	Trajectory control
DESCRIPTION:	Slow motor motion to stop
EXECUTION:	Immediate
CONDITIONAL TO:	A non zero
LIMITATIONS:	N/A
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
RELATED COMMANDS:	**G, S**
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The **X** command immediately abandons the current trajectory mode and causes the motor to slow to a stop using the current acceleration value **A**. This is different from the **S** command, which stops the motor a soon as possible without regard to the current acceleration. Regardless of the motion mode prior to the command, **X** leaves the motor position mode. The response to **RMODE** will be an "R".

EXAMPLE:

```
MP                  'Select Position Mode
A=200               'Set Acceleration
V=50000             'Set Velocity
P=1000000           'Set Position
G                   'Start Motion
WHILE Bt            'Loop while Trajectory
      IF UAI        'If input goes high
            X       'Decelerate now
      ENDIF
      RMODE         'response is "R"
LOOP
```

Related Command:

RUN

RUN?

APPLICATION:	Reset motor
DESCRIPTION:	Software reset motor to power up condition
EXECUTION:	Immediate
CONDITIONAL TO:	Serial character transmit completion
LIMITATIONS:	None
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
DEFAULT VALUE:	N/A
FIRMWARE VERSIONS:	ALL

DETAILED DESCRIPTION:

The **Z** command will totally reset the SmartMotor™ just as if power were taken away and later restored. Consequently, if there is a stored program, it will be run from the beginning. All modes of operation, variables and status bits will be restored back to their defaults. Subsequent to a power up or reset, the SmartMotor will

This command should not be used in a stored SmartMotor™ program.

1. initialize the motion mode, status bits and variables,
2. hold the serial port closed for approximately ¼ second
3. open and initialize the serial port
4. delay for ½ second. At the end of this time, the SmartMotor will examine the communications buffer. In versions 4.0 through 4.12, if any character is in the buffer, the stored program will not be executed. In versions 4.15 and later, the stored program will be aborted only if the specific characters "EE" are found.
5. The stored program will now run, unless aborted as described above.

After a program download, using the **Z** command is a very good way to evaluate how your SmartMotor™ will operate when powered on. The **RUN** command will execute the stored program, but it will not clear the motor to its default condition, so the subsequent operation will not necessarily mimic what would happen at power up.

WARNING! The Z command should not be used at or near the top of program code. In doing so, it may cause a continuous and repetitive resetting of the CPU and lock out the motor. IF this does happen, the Communications Lockup recovery tool may be used to regain access to the motor.

Za

Reset Peak Over Current Flag

Related Command:

Ba

RBa

APPLICATION: Program execution control

DESCRIPTION: Reset current limit violation latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **RBa**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

RESET VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Za resets the overcurrent error flag **Ba** to zero. If the current violation still exists **Ba** will be set to **1** again.

In early firmware versions, **Ba** was vallid only after being enabled by a **Za** or **ZS** command after the motion had started. This proved cumbersome to users, so enabling is not required in versions 4.15, 4.41, 4.75 and later. If **Ba** flag is regularly found to be set there may be a problem. Please verify the motor is correctly "sized" for the presently assigned task.

EXAMPLE:

```
IF Ba                           'Test flag
      PRINT("Over Current")
      Za                        'Reset flag
ENDIF
WAIT=4000
IF Ba                           'Retest flag
      PRINT("Over Current still in effect")
ENDIF
```

Related Command:

Bb

RBb

CHN0

CHN1

APPLICATION: Program execution control

DESCRIPTION: Reset serial data parity violation latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **RBb**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

RESET VALUE: 0

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Zb resets system flag **Bb**, the parity error violation latch, to zero. A parity error indicates that the communications has failed at a fundamental level. For safe operation, it is vital to find and eliminate the cause if this error flag is ever set.

EXAMPLE:

```
IF Bb                               'Test flag
      PRINT(" Parity Error ")
      Zb                            'Reset flag
ENDIF
```

Related Command:

Bc

RBc

APPLICATION:	Program execution control
DESCRIPTION:	Reset communications buffer overflow latch
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LANGUAGE ACCESS:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	**RBc**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

Zc resets system flag **Bc**, the serial communication receive buffer overflow violation latch, to zero. If the communication buffer overflows, the SmartMotor™ may receive a garbled or partial data byte. For safe operation, it is vital to find and eliminate the cause if this error flag is ever set.

EXAMPLE:

****

```
IF Bc                       'Test flag
    PRINT("Buffer Overflow")
    Zc                      'Reset flag
ENDIF
```

Zd
Reset Math Overflow Error Flag

Related Command:

Bd

RBd

APPLICATION: Program execution control

DESCRIPTION: Reset math overflow violation latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LANGUAGE ACCESS: N/A

LIMITATIONS: N/A

REPORT COMMAND: **RBd**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

RESET VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Zd resets the math overflow violation flag **Bd** to zero. For safe operation, it is vital to find and eliminate the cause if this error flag is ever set.

EXAMPLE:

```
IF Bd                           'Test flag
      PRINT("Math Overflow")
      Zd                        'Reset flag
ENDIF
```

Related Command:

Bd

RBd

APPLICATION:	Program execution control
DESCRIPTION:	Reset Position Error Status Bit "Be"
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LANGUAGE ACCESS:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	**RBd**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	**0**
FIRMWARE VERSIONS:	4.46 and higher

DETAILED DESCRIPTION:

Ze resets the Be Following error or position error flag to zero. This only works with PLS. PS2 and =4.76 firmware

EXAMPLE:

```
IF Be                         'Test flag
      PRINT("Following Error")
      Ze                      'Reset flag
ENDIF
```

Reset Comms Framing Error Flag

Related Command:

Bf

RBf

APPLICATION:	Program execution control
DESCRIPTION:	Reset serial communication framing error latch
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	**RBf**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

Zf resets system flag **Bf**, the serial communications framing error violation latch, to zero. A framing error means that the serial communications has failed at a fundamental level. For safe operation, it is vital to find and eliminate the cause if this error flag is ever set.

EXAMPLE:

```
IF Bf                       'Test flag
      PRINT("Framing Error")
      Zf                    'Reset flag
ENDIF
```

Related Command:

BI

RBI

APPLICATION: Program execution control

DESCRIPTION: Reset historical left limit latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **RBI**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

RESET VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

ZI resets system flag **BI**, the left limit latch, to zero. If you use **BI** to detect the activation of the left limit, take care to reset it with **ZI** before scanning for the bit again.

EXAMPLE:

```
IF Bl                                'Test flag
      PRINT("Left Limit Latched ")
      Zl                             'Reset flag
ENDIF
```

Zr

Reset Historical Right Travel Limit Flag

Related Command:

Br

RBr

APPLICATION: Program execution control

DESCRIPTION: Reset historical right limit latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **RBr**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

RESET VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Zr resets system flag **Br**, the right limit latch, to zero. If you use **Br** to detect the activation of the right limit, be sure to reset it with **Zr** before scanning for the bit again.

EXAMPLE:

```
IF Br                              'Test flag
      PRINT("Right Limit Latched")
      Zr                           'Reset flag
ENDIF
```

Reset Command Syntax Error Flag

Related Command:

Bs

RBs

APPLICATION:	Program execution control
DESCRIPTION:	Reset command scan error latch
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	N/A
REPORT COMMAND:	**RBs**
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUE:	**0**
FIRMWARE VERSIONS:	4.00 and higher

DETAILED DESCRIPTION:

Zs resets system flag **Bs**, the syntax or index access error latch, to zero. The **RBs** report and **ZS** commands may assist in discovering whether or not the present firmware version recognizes what appears to be a perfectly valid command and data packet.

EXAMPLE:

```
IF Bs                               'Test flag
      PRINT("Syntax Error")
      Zs                            'Reset flag
ENDIF
```

Zu

Reset Array Index Error state Flag

Related Command:

Bu

RBu

APPLICATION: Program execution control

DESCRIPTION: Reset user array index read access error latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **RBu**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

RESET VALUE: **0**

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Zu resets system flag **Bu**, the index read access violation latch, to zero. If the **Bu** flag is set, it means that you are improperly using an array and you may be writing data to an unspecified location. For safe operation, it is vital to find and eliminate the cause if this error flag is ever set.

EXAMPLE:

```
IF Bu                           'Test flag
      PRINT("Array Error")
      Zu                        'Reset flag
ENDIF
```

Related Command:

Bw

RBw

APPLICATION: Program execution control

DESCRIPTION: Reset encoder wrap around event latch

EXECUTION: Immediate

CONDITIONAL TO: N/A

LIMITATIONS: N/A

REPORT COMMAND: **RBw**

READ/WRITE: N/A

LANGUAGE ACCESS: N/A

UNITS: N/A

RANGE OF VALUES: N/A

TYPICAL VALUES: N/A

RESET VALUE: **0**

RELATED COMMANDS: Bw, RBw

FIRMWARE VERSIONS: 4.00 and higher

DETAILED DESCRIPTION:

Zw resets system flag **Bw**, the encoder wrap around violation latch, to zero. The SmartMotor™ tracks its position as 32 bit data, so a valid position is between **-2147483648** and **+2147483648**. If the motor moves out of this range, the position will overflow or "wrap around". It is therefore advisable to not operate any following mode, cam mode, absolute position move, or relative position move such that wrap around may occur. Reset the origin to avoid operating in this region.

EXAMPLE:

```
IF Bw                              'Test flag
      PRINT("Wraparound Occurred")
      Zw                           'Reset flag
ENDIF
```

Related Command:

Za

Zb

Zc

Zd

Zf

Zl

Zr

Zs

Zu

Zw

APPLICATION:	Program execution control
DESCRIPTION:	Reset software system latches to power up state
EXECUTION:	Immediate
CONDITIONAL TO:	N/A
LIMITATIONS:	None
REPORT COMMAND:	N/A
READ/WRITE:	N/A
LANGUAGE ACCESS:	N/A
UNITS:	N/A
RANGE OF VALUES:	N/A
TYPICAL VALUES:	N/A
RESET VALUES:	N/A
FIRMWARE VERSIONS:	4.00 and higher, 4.76 and higher, see below

DETAILED DESCRIPTION:

Almost any event that occurs within a SmartMotor™ gets recorded in system flags. These flags can be read as part of a program or a host inquiry. Once read, it is necessary to reset the flag that records the particular event in order to record the next occurrence. **ZS** resets all of the latched bits in the **S** status byte and the **W** status word, as well as the three communication status bits: **Ba**, **Bb**, **Bc**, **Bd, Be**, **Bf**, **Bl**, **Br**, **Bs**, **Bu** and **Bw**.

ZS performs the following flag resets:

Za	Reset hardware current limit violation
Zb	Reset serial data parity error
Zc	Reset communications buffer overflow
Zd	Reset user math overflow
Ze	Reset Position Error (In >=4.76 firmware only.)
Zf	Reset communications framing error
Zl	Reset historical left limit
Zr	Reset historical right limit
Zs	Reset user command syntax error
Zu	Reset user read array indexing out of range
Zw	Reset wraparound

CONTINUED ON NEXT PAGE

Related Command:

Za

Zb

Zc

Zd

Zf

Zl

Zr

Zs

Zu

Zw

EXAMPLE:

```
        ZS              'reset all error and limit flag latches
                        'useful for debugging new programs
                        'but not satisfactory for real time control
                        'consider the following
        C900            'Error Report Subroutine
        IF Ba           'Test flag
                PRINT("Over Current")
        ENDIF
        IF Bb           'Test flag
                PRINT("Parity Error")
        ENDIF
        IF Bc           'Test flag
                PRINT("Buffer Overflow")
        ENDIF
        IF Bd           'Test flag
                PRINT("Math Overflow")
        ENDIF
        IF Bf           'Test flag
                PRINT("Framing Error")
        ENDIF
        IF Bl           'Test flag
                PRINT("Left Limit")
        ENDIF
        IF Br           'Test flag
                PRINT("Right Limit")
        ENDIF
        IF Bs           'Test flag
                PRINT("Syntax Error")
        ENDIF
        IF Bu           'Test flag
                PRINT("Array Error")
        ENDIF
        IF Bw           'Test flag
                PRINT("Wraparound Occurred")
        ENDIF
        ZS              'Reset all tested flags. Faulty !!!

END             'By the time ZS is executed it is possible,
                'some previously tested zero flags may now be set.
```

Array Variable Memory Map

aa	al[0]	aw[0], aw[1]	ab[0], ab[1], ab[2], ab[3]
bb	al[1]	aw[2], aw[3]	ab[4], ab[5], ab[6], ab[7]
cc	al[2]	aw[4], aw[5]	ab[8], ab[9], ab[10], ab[11]
dd	al[3]	aw[6], aw[7]	ab[12], ab[13], ab[14], ab[15]
ee	al[4]	aw[8], aw[9]	ab[16], ab[17], ab[18], ab[19]
ff	al[5]	aw[10], aw[11]	ab[20], ab[21], ab[22], ab[23]
gg	al[6]	aw[12], aw[13]	ab[24], ab[25], ab[26], ab[27]
hh	al[7]	aw[14], aw[15]	ab[28], ab[29], ab[30], ab[31]
ii	al[8]	aw[16], aw[17]	ab[32], ab[33], ab[34], ab[35]
jj	al[9]	aw[18], aw[19]	ab[36], ab[37], ab[38], ab[39]
kk	al[10]	aw[20], aw[21]	ab[40], ab[41], ab[42], ab[43]
ll	al[11]	aw[22], aw[23]	ab[44], ab[45], ab[46], ab[47]
mm	al[12]	aw[24], aw[25]	ab[48], ab[49], ab[50], ab[51]
nn	al[13]	aw[26], aw[27]	ab[52], ab[53], ab[54], ab[55]
oo	al[14]	aw[28], aw[29]	ab[56], ab[57], ab[58], ab[59]
pp	al[15]	aw[30], aw[31]	ab[60], ab[61], ab[62], ab[63]
qq	al[16]	aw[32], aw[33]	ab[64], ab[65], ab[66], ab[67]
rr	al[17]	aw[34], aw[35]	ab[68], ab[69], ab[70], ab[71]
ss	al[18]	aw[36], aw[37]	ab[72], ab[73], ab[74], ab[75]
tt	al[19]	aw[38], aw[39]	ab[76], ab[77], ab[78], ab[79]
uu	al[20]	aw[40], aw[41]	ab[80], ab[81], ab[82], ab[83]
vv	al[21]	aw[42], aw[43]	ab[84], ab[85], ab[86], ab[87]
ww	al[22]	aw[44], aw[45]	ab[88], ab[89], ab[90], ab[91]
xx	al[23]	aw[46], aw[47]	ab[92], ab[93], ab[94], ab[95]
yy	al[24]	aw[48], aw[49]	ab[96], ab[97], ab[98], ab[99]
zz	al[25]	aw[50], aw[51]	ab[100], ab[101], ab[102], ab[103]
aaa	al[26]	aw[52], aw[53]	ab[104], ab[105], ab[106], ab[107]
bbb	al[27]	aw[54], aw[55]	ab[108], ab[109], ab[110], ab[111]

Variable	Long (MSB–LSB)	Word (MSB–LSB)	Byte (MSB–LSB)
ccc	al[28]	aw[56]	ab[112]
			ab[113]
		aw[57]	ab[114]
			ab[115]
ddd	al[29]	aw[58]	ab[116]
			ab[117]
		aw[59]	ab[118]
			ab[119]
eee	al[30]	aw[60]	ab[120]
			ab[121]
		aw[61]	ab[122]
			ab[123]
fff	al[31]	aw[62]	ab[124]
			ab[125]
		aw[63]	ab[126]
			ab[127]
ggg	al[32]	aw[64]	ab[128]
			ab[129]
		aw[65]	ab[130]
			ab[131]
hhh	al[33]	aw[66]	ab[132]
			ab[133]
		aw[67]	ab[134]
			ab[135]
iii	al[34]	aw[68]	ab[136]
			ab[137]
		aw[69]	ab[138]
			ab[139]
jjj	al[35]	aw[70]	ab[140]
			ab[141]
		aw[71]	ab[142]
			ab[143]
kkk	al[36]	aw[72]	ab[144]
			ab[145]
		aw[73]	ab[146]
			ab[147]
lll	al[37]	aw[74]	ab[148]
			ab[149]
		aw[75]	ab[150]
			ab[151]
mmm	al[38]	aw[76]	ab[152]
			ab[153]
		aw[77]	ab[154]
			ab[155]
nnn	al[39]	aw[78]	ab[156]
			ab[157]
		aw[79]	ab[158]
			ab[159]
ooo	al[40]	aw[80]	ab[160]
			ab[161]
		aw[81]	ab[162]
			ab[163]
ppp	al[41]	aw[82]	ab[164]
			ab[165]
		aw[83]	ab[166]
			ab[167]
qqq	al[42]	aw[84]	ab[168]
			ab[169]
		aw[85]	ab[170]
			ab[171]
rrr	al[43]	aw[86]	ab[172]
			ab[173]
		aw[87]	ab[174]
			ab[175]
sss	al[44]	aw[88]	ab[176]
			ab[177]
		aw[89]	ab[178]
			ab[179]
ttt	al[45]	aw[90]	ab[180]
			ab[181]
		aw[91]	ab[182]
			ab[183]
uuu	al[46]	aw[92]	ab[184]
			ab[185]
		aw[93]	ab[186]
			ab[187]
vvv	al[47]	aw[94]	ab[188]
			ab[189]
		aw[95]	ab[190]
			ab[191]
www	al[48]	aw[96]	ab[192]
			ab[193]
		aw[97]	ab[194]
			ab[195]
xxx	al[49]	aw[98]	ab[196]
			ab[197]
		aw[99]	ab[198]
			ab[199]
yyy	al[50]	aw[100]	ab[200]
			ab[201]
		aw[101]	ab[202]
			ab[203]
zzz	Note: The "zzz" memory location is used for SWITCH-CASE calculations. Do not use it if the SWITCH command is being used in user code.		