

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

MotionBASIC
Hypertext Manual

Hard Copy Edition
GN3-MBHj

Copyright (c) 1996

Ormec Systems Corp.

All rights reserved.

19 Linden Park

Rochester, NY 14625

(716) 385-3520

January 19, 1996

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionBASIC Reference Manual (MB v3.2a)

Revision: 32.1

Introduction	Introduction to this Manual
MotionBASIC Overview	Overview of the MotionBASIC Programming Language
Motion Controllers	ORMEC's ORION Motion Controllers
MotionPRO	ORMEC's MotionBASIC Program Development Software
Hypertext Keys	Hypertext Function Key Description
Commands	MotionBASIC Commands
Statements	MotionBASIC Statements
Functions	MotionBASIC Functions
Operators	MotionBASIC Operators
Constants	Use of constants in MotionBASIC
Variables	Variable Types supported by MotionBASIC
ORMEC Variables	ORMEC Pre-defined Variables for Motion Control
Error Codes	Description of MotionBASIC error codes
Troubleshooting	Troubleshooting guide & System Module LEDs
ASCII Table	Table showing all ASCII codes in decimal and hex
Keywords	Alphabetical List of all Keywords and Statements

Copyright (c) 1989-1996, Ormec Systems Corp., All Rights Reserved

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

- MotionBASIC is a ROM based interpreter built on a multi-tasking operating system. Motion control, I/O, timing functions, **event processing** and error processing are fully serviced.

- the **set** data-type and a number of set **operators** have been added to allow powerful and easy-to-use multi-axis motion programming;

- the user program can automatically execute on **powerup** in addition to operating in the modes: **direct** and **indirect**.

- user program variables (except **string variables**) are maintained thru power-down in **non-volatile memory**, see: **Program Initialization**

- MotionBASIC functionality can be extended with special device drivers or with additional statements and functions via **MotionCARDs**.

MotionBASIC program development is supported by **MotionPRO**, which runs on an IBM-PC or compatible.

Keywords
MotionBASIC **keywords**, such as IF, THEN, GOTO, MOVE and RETURN, are integral to statements or commands. In addition to BASIC keywords, MotionBASIC includes keywords that create motion and that refer to the pre-defined **ORMEC variables**. **Keywords** are also called **reserved words**. They cannot be used as user variables or labels. They are stored in memory as tokens (of 1 or 2-bytes) for efficient memory usage.

Commands
Commands are executable instructions in MotionBASIC which are primarily used for writing, editing, or running programs. Most commands are used in **direct mode**.

Statements
Statements are executable instructions in MotionBASIC which cause it to perform a desired action such as "perform a computation" or "move a servomotor". Statements can be executed in **direct mode**, but they are primarily put together in a logical sequence to create programs.

Motion Statements are instructions which create servo motion. Included are **MOVE**, **GEAR**, and **HALT**.

"Programs" are collections of statements which are stored in **Program Files** which are loaded into controller memory to be executed in **indirect mode**.

Program statements must conform to a standard **line format**. The MotionBASIC **line editor** is used to enter or modify MotionBASIC program statements in controller memory.

MotionPRO development software includes a **text editor** which is used to edit programs stored in **Program Files** on **disk**.

Constants, Variables & Expressions
Constants are static values which MotionBASIC uses during execution of a program. **Variables** are names which represent changeable values used in a program. Most variables have their names chosen by the

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Differences from GWBASIC

- Multi-character relational operators must be specified in the proper sequence without spaces. Use the documented operators.

- Trailing spaces up to the next comma or end of line are not truncated.

- The apostrophe character (') always starts a comment unless it is inside quotation marks. Therefore the DATA statement

  10 DATA Motor's Speed in GWBASIC

  must be written

  10 DATA "Motor's Speed" in MotionBASIC

- Branching out of a **subroutine**, a **FOR**, **NEXT**, or a **WHILE**, **WEND** loop (using **GOTO**) leaves old information on the control stack. If done repeatedly without executing a **STACK** CLEAR statement, an "Out of Memory" error will occur.

- The string value returned by the function **STR$** does not include a leading space, even if the number is positive.

- The **COLOR** statement does not allow background color numbers above 7.

- Operator **precedence** is for the unary negation (-) operator to be higher than the exponentiation (^) operator.

- The **RUN** command does not clear user program variables. User variables (other than **string variables**) are maintained across power cycles in **non-volatile memory**, as described in the **program initialization** chapter.

- The **CLEAR** command does not reset error trapping established by **ON ERROR**.

- The KILL command will not generate an error when the file to be deleted is non-existent.

- The **RMDIR** command will not generate an error when the directory to be deleted is non-existent.

- The **CHAIN** command will clear program variables if the MERGE option is not specified.

In additional to the above, the following highlights most of the incompatibilities in Random Access File support between MotionBASIC and GWBASIC.

- MotionBASIC does not support simultaneous **FIELD** statements like GWBASIC does. A FIELD statement for the same file number in MotionBASIC overwrites the previous one.

- MotionBASIC supports direct input/output of binary information while GWBASIC does that through the functions (MKI$, MKD$, MKS$, CVI, CVD, CVS).

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Program Labels

MotionBASIC implements program labels to allow you to symbolically mark reference points in your programs. Labels are composed of any number of contiguous alphanumeric characters & "periods" [.], starting with an alpha character and followed by a colon [:]. e.g. **LABEL.NAME:** All labels must be the first item on a program line. References to a label should **not** include the **:**.

MotionBASIC performs an **implied GOSUB** when a label is used by itself as a command or statement. This is the same as a **GOSUB** to that label. e.g. You can execute the **M5** subroutine below by typing **M5** and pressing the Enter key. To distinguish implied GOSUBs from labels in programs, always follow them with a space if they're not at the end of a line.

Example:

```
 10 M5:              'establish a label entitled M5
 20 MODE@ = 5        'set mode@ to position mode
 30 RETURN
524 M5 :PRINT "Position Mode" 'call the M5 subroutine (implied GOSUB)
```

MotionBASIC Powerup Operation

If the red **RESET** button is pressed, or power is cycled, Generation III controllers perform on-line diagnostics, & check the black **PB1** button.

If PB1 **is held in**, the controller checks for a **Memory Disk**, and for a program file on it named, "**PB1LOAD.BAS**". If found, this file is loaded into **non-volatile memory**. Program execution is inhibited.

If PB1 **is not held in**, the controller checks for a **Memory Disk**, and for a program file on it named, "**AUTOLOAD.BAS**". If found, this file is loaded into **non-volatile memory**. Whether or not a file was loaded, MotionBASIC then checks non-volatile memory contents for a line number **0**. If found, **indirect** program execution is initiated at line 0.

User functions & **variables**, other than **strings**, are non-volatile, as described in the **Program Initialization** section.

While a program file is being "loaded" from a **Memory Disk**, the **User 1** LED will flash to indicate that the load is occurring.

Serial communications can be initiated with the controller by sending two **ASCII Carriage Return** characters. MotionBASIC will print a "Sign-on message" and either continue **indirect** operation or go to **direct** mode.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionBASIC Program Initialization

Powerup, **RUN** and **CLEAR** have the effects described in the table below:

	Powerup	**RUN**	**CLEAR**	**CHAIN**
Array dimensions	---	---	RESET	---
String space size	---	---	DEFAULT	---
String variables	RESET	---	RESET	---
Non-string User **variables**	---	---	RESET	---
User **data types**	---	---	**INTEGER**	---
ORMEC **axis variables**	DEFAULT	DEFAULT	DEFAULT	---
ORMEC **unit parameters**	DEFAULT	---	---	---
Key definitions	RESET	RESET	---	RESET
ON EVENT, **ON ERROR**, etc.	RESET	RESET	---	RESET
Ports	**CLOSE**	CLOSE	CLOSE	---
User Functions	---	---	RESET	---
TRACE & **BREAK**	RESET	---	---	---
TRACE VAL	RESET	---	RESET	---
ERR & **ERL**	---	---	RESET	---
Abs Encoder quad feedback	DISABLE	DISABLE	DISABLE	---

Line Format

Program statements in MotionBASIC must conform to the following line format conventions:

- must begin with a line number (or a line number preceded by spaces)
- the line number may be any integer between 0 and 65529
- each numbered line can have between 1 and 127 characters
- each line may contain multiple statements, with each statement separated by a colon (:)
- lines of MotionBASIC will be executed in numerical order, with multi-statement lines executed from left to right unless otherwise directed by an individual MotionBASIC statement.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Expressions

Expressions may be simply string or numeric constants, or they may combine **constants** and **variables** with **operators** and **functions** to produce a single value. Automatic **type conversion** will be performed during expression analysis if appropriate.

The term *numexpr* is used in the function and statement descriptions to refer to any numeric expression. It can be of any numeric **data type**, including **integer**, **long integer**, and both **single** and **double precision** floating point. It can also be **set** although it may not make much sense in most situations.

The term *intexpr* is used to refer to numeric expressions which either are, or will be converted to, **integer** by the statement or function. Any fractional part of *intexpr* will be rounded, and expressions which evaluate to values out of the integer range will cause a **syntax error**.

The term *lngexpr* is used to refer to numeric expressions which either are, or will be converted to, **integer** or **long integer** by the

statement or function. If the expression starts out as **single** or **double** precision floating point, MotionBASIC will first attempt to convert to **integer**, and if not possible, will attempt to convert to **long integer**. Any fractional part of *lngexpr* will be rounded, and expressions which evaluate to values out of the long integer range will cause a **syntax error**.

The term *setexpr* is used to refer to numeric expressions which either are, or will be converted to, **set** by the statement or function. If the expression starts out as **single** or **double** precision floating point, a "type mismatch" **expression error** will be generated.

Generation III Motion Controllers

MotionBASIC is the language used to program ORMEC's Generation III family of multi-axis motion controllers. These controllers utilize a multi-processor architecture combining Intel's premier line of microprocessors with Texas Instruments' DSPs through a shared-memory interface that brings unprecedented computer power to multi-axis electronic motion control.

Two Generation III Motion Controllers are currently available.

The **Model 20** has 2 expansion slots providing the capability to control as many as 4 servo motors.

The **Model 40** has 4 expansion slots providing the capability to control as many as 8 servo motors.

Both the **Model 20** and the **Model 40** have a 12 MHz Intel 80C186 main processor, an optional 80C187 math co-processor, and one megabit of internal **non-volatile memory** standard.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Model 40 Motion Controller

Operator Interfaces **Available:**
- EGA Color or Monochrome
  Industrial Monitor with Keypad
- Serial Industrial Terminal
  8x40 LCD, alphanumeric 30 key pad
- Multi-drop Industrial Terminals
  1x16 LCD, numeric 24 key pad
- Multi-drop Displays
  Vacuum Fluorescent
- IBM-PC compatible keyboard
- VGA, EGA, or Mono Monitor

Expansion Bus:
- RS-232 and/or RS-422 serial communications
- Up to 64 serial devices
- Interface to computers or PLCs

Inputs/Outputs:
- 16 discrete I/O points
- **Option**: Extended I/O Bd. for:
  - an I/O expansion: may add 24 additional digital I/O
    OR
  - an I/O expansion: may add 512 analog or digital I/O with Opto-22's **Pamux**

Indicators:
- 16 Digital I/O LEDs
- 20 **System Status LEDs**
- **E-STOP OK** LED
- **NO FAULT** LED
- **No Fault Relay** output

MotionNET:
- RS-485 Communications Port
- 1.5 M bits per sec synchronous

MotionDATA:
- **Electronic gearing** - linear or **cascaded control**
- 612.5 KHz DSP to DSP pipeline

AC Brushless Servomotors:
- Peak torques from 2 to 930 lb-in
- Speeds to 4,500 RPM
- Encoder resolutions up to 24,000 counts/rev
- Accelerations to 57,000 rad/sec^2

Industrially Hardened
- Switching Power Supply
- Incoming Line Filter
- **Watchdog Timer** Circuitry
- Integrated **Emergency Stop**
- **No Fault Relay**
- Extensive **diagnostics**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

ORMEC Operator Interface Devices

In an ORMEC Generation III control system, there are three types of operator interface devices supported in MotionBASIC, as standard.

The **OIT, Operator Interface Terminal**, is a rugged industrial video monitor w/ keyboard. Available as a color or monochrome monitor. The **OIT Display** has a **25 line by 80 column** screen. It has a membrane style keypad for numeric data entry. **OIT Function Keys**, F1 thru F10, are standard and fully supported by MotionBASIC.

The **MMI-840, Man-Machine Interface**, is a rugged industrial terminal that is a serial device. The **MMI Display** is an **8 line by 40 column** backlit LCD screen. A rubberized keypad provides for alphanumeric data entry as standard, and the **MMI Function Keys**, F1 thru F8, which are fully supported by MotionBASIC.

The **ITM-270, Industrial Terminal**, is a rugged serial device with a numeric keypad with a **1 line by 16 character** display (buffered to 80 characters). It has 6 Function Keys, see **ITM Display & Function Keys**.

OIT Display & Function Keys

DISPLAY:

The **OIT** Display has 25 display lines (rows) and 80 display columns.

- line (row) 25 is "off-limits" while **function key text** is displayed, (prohibits **LOCATE**, **PRINT**, & **INPUT @** on row 25)

FUNCTION KEYS:

The OIT has 10 function keys, F1 thru F10.
The following statements apply to their use:

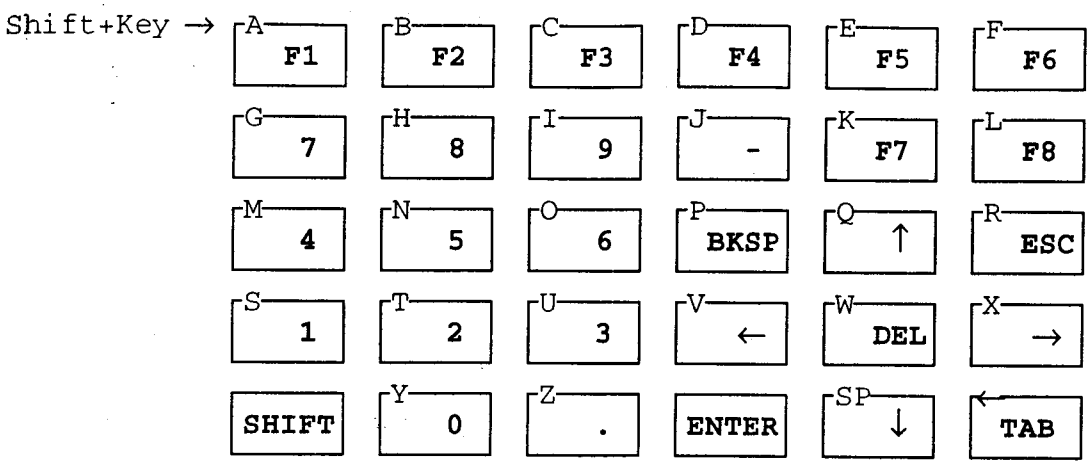
KEY	Syntax 1, Assign **function key text** labels
KEY	Syntax 2, Turn ON/OFF the function key label display
KEY	Syntax 3, Assign additional keys for special uses
ON KEY...GOSUB	Establish subroutines for individual keys
KEY (n)	Enable trapping of these keys, and thus the ability to perform the established subroutine.

- Each OIT function **KEY**'s text label is displayed to 6 characters.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MMI Keypad Layout

The MMI has rubberized keys that provide tactile feedback for alpha-numeric data entry. The keypad layout is as follows:



Function Key Text & Extended Function Key Text

With **operator interface devices** it is often desirable to use function keys to select menu options. MotionBASIC supports function key labels that automatically prompt the operator. See: **KEY** statement.

Function key label text is truncated to 15 characters. Thus **KEY LIST** will report current function key text only up to 15 characters.

The **OIT Display** provides only the bottom row of the display for function key labels. It displays only the first 6 of the 15 characters defined per label.

The **MMI Display** can provide the bottom **two rows** for function key labels. When the length of any one key's text label exceeds that key's base label width, the text string is divided and the bottom two rows are used for function key label display.

See Also: **MMI Function Key Text**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Memory Conservation Techniques

When writing very large programs in MotionBASIC, the user may encounter the error statement, **#1801** Out of Memory. This error occurs when a program, and its variables, becomes too large to fit into the available memory space. See: **Non-Volatile Memory**. Using the following techniques will reduce the amount of memory that a program requires.

- Eliminate leading spaces. Internally, memory conservation is accomplished whenever spaces are consecutively repeated more than three times. However, even with this feature, two bytes of memory are consumed for each set of repeated spaces.

- Use multiple statements per **line**. There is a five byte overhead for each numbered line in a program.

- Use **subroutines**. This avoids repeating sections of code that need only be written once.

- Eliminate downloaded program **comments**. This will provide additional

  memory. Upto 20 Kbytes of comment text, is stored in a dedicated and separate section of memory, See: **Memory Map**. User program space is slightly reduced since only three bytes of memory per comment is consumed, regardless of the comment length. If the total comment text exceeds the 20K capacity, the "overflow" will reside entirely in user program space. **Disk Comments**, are an excellent alternative to downloaded comments, since they do not consume any controller memory (or **Memory Disk** space if the file has a ".BAS" extension).

- Use a **variable** rather than repeat a numeric literal **constant** within a program. For example, using the constant 3.14159265 ten times in a program consumes much more memory than defining PI# = 3.14159265, and then using the variable PI# repeatedly in a program.

- Wherever possible, use **integers** instead of **single precision** reals. And where possible use **single precision** instead of **double precision** reals. See: **Memory Usage for Variables**

- Avoid defining lengthy variable names. Each character in the name of user defined variables consumes one byte of memory.

- Reuse variables. For example, variables which hold temporary results in one part of a program can be used again for the same purpose in other parts of a program. Also, if a program has many separate **FOR...NEXT** loops, the same index may be used in each loop.

- **ERASE** unused variables. Memory that was allocated to a particular variable can be reused if that old variable is erased.

- A program does not need to have an **END** statement. This will save seven bytes.

- Remember to use the zero elements of **array** variables. If you need ten elements in an array, then only dimension it for a maximum index of 9, **DIM** A(9).

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionCARDs

MotionCARDs plug into the **internal card slot** of a Generation III motion controller and are programmed at ORMEC to extend the functionality of MotionBASIC. Standard MotionCARDs currently support:

Modbus	Statements are provided which allow the user to "map" program variables to register numbers. A background task automatically transfers register data serially using this factory communications network protocol.
Data Highway/Plus	"Mapped" register data is transferred by a co-processor adapter at 57.6K baud using this peer-to-peer factory network protocol.
S908/Modbus	This MotionCARD allows a Generation III motion controller to be configured as an "S908 remote I/O drop" and also transfers 16 or more integer variables in one PLC scan.
Delay Counters	This MotionCARD provides MotionBASIC support for the optional **DSP** delay counters.

In addition, custom MotionCARDs can be developed at ORMEC to accomplish specific application objectives by adding functionality to MotionBASIC.

Important: Please note that the program buffer must be completely empty when the MotionCARD is installed. To make sure that this is the case set **PW@**=3 and type **NEW**. Be sure to save your program first!

Important: The power to the Gen III **must** be off when installing or removing a MotionCARD.

DSP Axis Modules

Generation III **DSP Axis Modules** utilize the latest Digital Signal Processor technology to implement all digital high performance motion control. Each DSP communicates with the main processor through an on-board **shared memory interface.** It implements one or two **servo control loops** utilizing discrete signal processing algorithms which update the loops at up to 5,000 times per second eliminating the need for analog circuitry.

In addition to a wide variety of advanced motion control software features, DSP Axis Modules also provide:

- individual interfaces to each servodrive and/or **axis encoder;**
- **electronic gearing** via the **MotionDATA** communications link;
- an optional delay counter (**DELAY@**) per axis; and
- Inputs & Outputs interfaced via a **DSP I/O Terminal Block:**
  two optional 12-bit analog inputs (**AIN1@** & **AIN2@**) per axis.
  two **hardware overtravel limit** inputs per axis
  three **electronic limit switch** outputs per axis (TTL compatible)
  four high speed inputs (**ASEN@**, **BSEN@** & two **ZREF@**s),

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

DSP Axis Module Memory

Generation III **DSP Axis Modules** have two on-board memory resources:

- The 2K byte dual-port *shared memory* is used for real-time communications between the main processor and the axis module. The main processor accesses this memory via the its IBM-PC/AT style bus and the DSP accesses it through its other port.

  The pre-defined **ORMEC variables** for the axes controlled by a particular DSP Axis Module are maintained in this memory. It also contains a **motion queue** which is a FIFO Register used by MotionBASIC to transmit motion command information to the **DSP Axis Module.**

- The 64K byte *local memory* is used for the DSP's local code and data requirements.

Motion Queue

Motion statements transfer motion command information to individual **DSP Axis Modules** via the **Motion Queue** which is a "First In, First Out" (FIFO) register located in the **shared memory interface.**

Several motions may be sent to the DSP, providing the following:

- MotionBASIC can pre-process motion command information, for later execution by the DSP;
- the DSP can execute queued motion independently and quickly, responding to high-speed sensors or a continuous stream of **pacer** information; and
- MotionBASIC can continue to execute and handle other tasks while the individual DSPs handle their motion queues.
- The queued motions can repetitively execute at the DSP. See: **REPEAT**

MOVE AT, **GEAR AT** and **HALT** commands generally clear the Motion Queue of other pending motion commands. Multiple **MOVE FOR**, **MOVE TO** and **GEAR FOR** commands can be stacked in the queue, and will also be superimposed on any currently executing **GEAR AT** command.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Kp - Position Loop Proportional Gain

When using **torque mode servos** Kp is set by the motion controller as a function of the velocity loop time constant and the gain factor variable **KP@**. With KP@ = 100%, the position loop is adjusted for a critically damped position response. Adjusting this factor higher results in an underdamped response and adjusting it lower results in an overdamped response. The standard setting of 100% meets the requirements of most applications.

When using **velocity mode servos** Kp is set by the user in units of in/min/mil. The position loop response for any particular setting is dependent on the velocity loop tuning in the servodrive.

Kpi - Position Loop Integral Gain

Kpi is set by the motion controller as a function of the position loop characteristics and the gain factor variable **KPI@**.

With KPI@ = 100% and **Kp** properly adjusted, the servo axis is adjusted for critically damped position response. Adjusting this factor higher results in an underdamped response and adjusting it lower results in no improvement in response. Standard settings of either 0 or 100% meet the requirements of most applications.

Position loop integral gain (**KPI@**) is normally required only in applications where position error during motion is critical, such as electronic gearing or in the control of continuous web systems. It may also be required to eliminate static position errors in applications with analog velocity loops, such as axes using **distributed feedback** or **velocity mode servodrives**.

Kv - Velocity Loop Proportional Gain

When using **torque mode servos** the velocity loop proportional gain is set by the motion controller to achieve the velocity loop time constant objective established by the **VLTC@** variable.

The performance characteristics of servo motion control systems are highly dependent on velocity loop time constant, and so **VLTC@** should be set as small as the system dynamics allow. A realistic goal is 0.7 to 3 msec for loads which are tightly coupled to the motor.

When using **velocity mode servos** the velocity loop gain and response is controlled by the servodrive and changes to **VLTC@** have no affect.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Kvh - Velocity Observer Feedback Gain

The velocity observer feedback gain (Kvh) is normally set to 100%, fully connecting the output of the **velocity observer** to the velocity loop summing junction.

To allow the use of **analog velocity feedback** or **velocity mode servos**, it is possible to adjust *Kvh* using the **KVH@** variable. This proportionally reduces the velocity feedback from the **velocity observer**, which derives its velocity information from the **axis encoder**.

See also: **Distributed Feedback Systems**

Kvha - Analog Velocity Feedback Gain

The analog velocity feedback gain (Kvha) is provided primarily to accommodate systems using **distributed feedback**. Generation III **DSP Axis Modules** can close an analog velocity loop using the optional **AIN2** 12-bit analog input. To use analog velocity feedback instead of the **velocity observer**, **install a DSP Axis Module with the A/D converter option**, and adjust Kvha using the **KVHA@** variable. To disable velocity observer feedback, set **KVH@** to 0.

ORMEC's servodrives provide analog velocity monitor signals derived from the encoder integral to the servomotor which can be used for this purpose. The standard axis interface cables connect this signal to the DSP Axis Module's AIN2 input for convenience.

If desired, it is possible to provide velocity loop control using a combination of analog and the **velocity observer** feedback. Simply reduce the setting for **KVHA@** and proportionately set **KVH@**. e.g. Multiply a **KVHA@** setting by 0.6 and set **KVH@** to 40 to obtain 60% of the velocity loop feedback from the analog input and 40% from the velocity observer.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Torque Mode Servos

Torque mode servos provide motor torque proportional to the servodrive's input voltage. Motor torque is proportional to armature current, and so torque mode servodrives are also called "current mode servodrives" because they actually control armature current.

Torque mode servodrives usually close current loops in the motor armature circuitry for accuracy and response, but it is important to note that **adjustments to the current loops are totally load independent.** There is therefore no reason for any user adjustments in a torque mode servodrive.

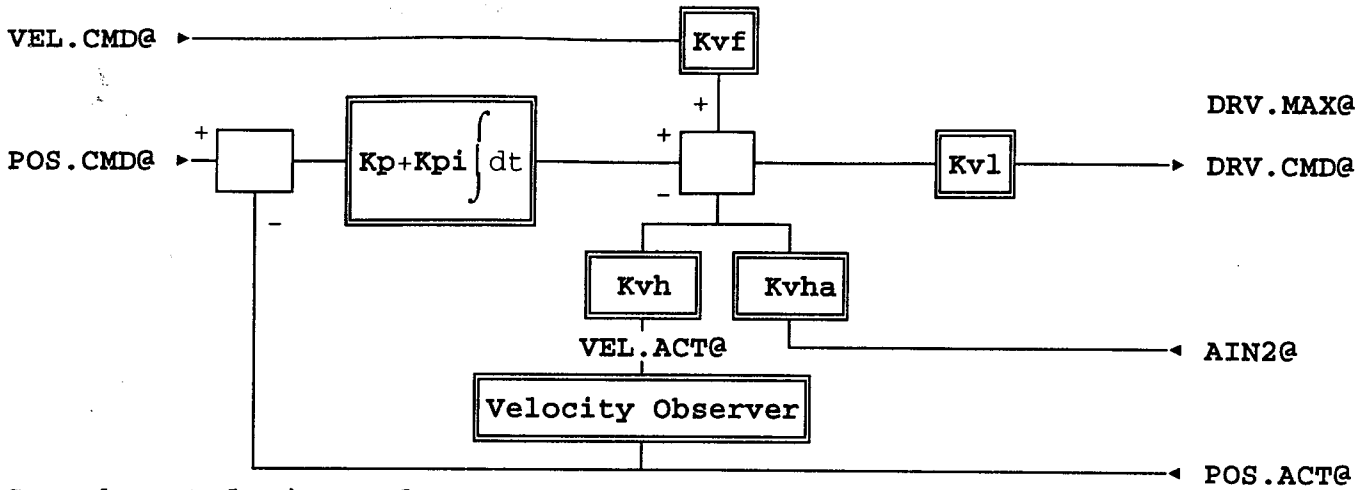
With torque mode servos, all load dependent loop adjustments are made in MotionBASIC. Since the DSP Axis Module's drive command output commands torque, it is also easy to both limit and monitor axis torque.

Torque mode servos are preferred in most applications.

See also: **Torque Mode Servo Loop Architecture** **Velocity Mode Servos**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Velocity Mode Servo Loop Architecture



See also: **Velocity Mode Servos** **VEL.GAIN@** **Loop Architecture**

Velocity Loop Output Gain

The velocity loop output gain (Kvl) translates standard velocity units used in the DSP to the proper drive command output from the DSP's digital to analog converter. It is automatically calculated from the specified **velocity mode servo gain**.

VEL.GAIN@ **Velocity Mode Servo Gain** ORMEC Variable

Purpose: Configure an axis as a **velocity mode servo** and define the gain of the velocity mode servodrive used with that axis.

Syntax: VEL.GAIN@[(***axes***)] = *factor*

usage: ***var%*** = VEL.GAIN@[(***axes***)]

factor An **integer** expression which represents a **velocity mode servo** gain in RPM/volt from **MTR.SPD.LIM@**/10 to 1500. The default of 0 configures the axis as a **torque mode servo**.

When configuring an axis as a **velocity mode servo**, **VEL.GAIN@** must be set first, followed by the other parameters in the **velocity mode** servo loop architecture.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Encoder Backup Compensation

Encoder Backup Compensation is used to precisely track reverse **pacer** travel when **GEAR** statements are executed. This is done because profiled geared motions are **ONLY** relevant to **forward pacer travel**, see: **CW.FWD@**. If the pacer "backs up", the **follower** must keep track of the total backup distance so that it can perform the profiled geared motion on forward pacer travel from the "initial gearing point".

A **GEAR AT** command may be treated differently. If a GEAR AT has zero **Fa** and **Pa** distance parameters, it is essentially a request for an immediate ratio change. Pacer motion, forward or reverse, is not needed to establish the new follower ratio.

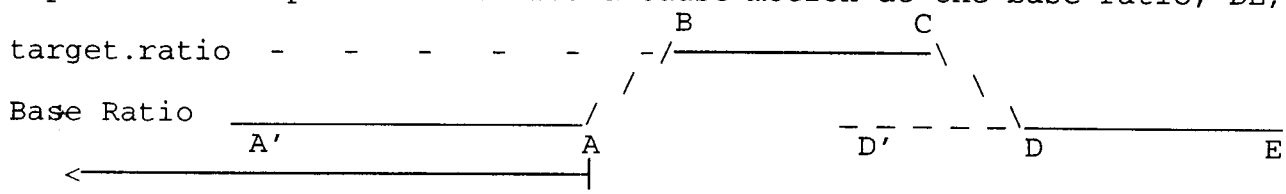
When **Fa** or **Pa** are non-zero (**Pa** > MIN Pa), then the GEAR AT statement in this case defines a "profiled gear motion". The follower will ramp to the new ratio on forward pacer travel ONLY. Therefore the EBC would be active to track any reverse pacer motion at the starting point of the ratio ramp.

The EBC can accumulate as many as 2^31 pacer counts. (2,147,483,647 cnts)

Example: A Profiled Gear Motion: GEAR **FOR a distance.**
Assume that the statement is issued when a follower is at point A, below:

GEAR follower **FOR** distance **AT** target.ratio

Forward pacer motion gears the follower into the segments AB, BC, CD.
Reverse pacer motion, before the GEAR FOR is complete at point D, would gear the follower back through the profile, segments DC, CB, BA.
Continued reverse pacer motion would be EBCed, segment AA'.
With forward travel, at point D, **DSP.DONE@** would be **TRUE**, and the GEAR FOR complete. Then pacer travel would cause motion at the base ratio, DE, DD'

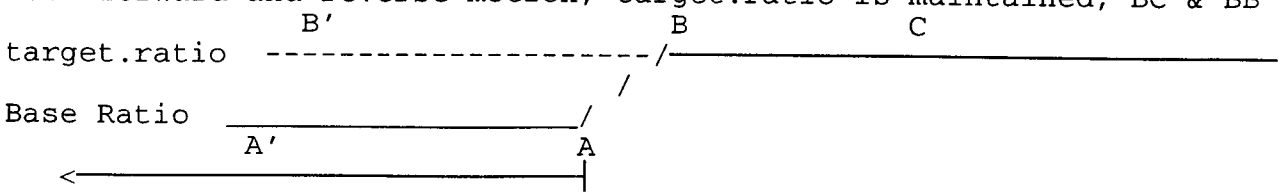


GEAR.RATIO@ shows current ratio

Example: A Profiled Gear Motion: GEAR **AT a ratio IN** (or BY) **some distance**
Assume that the statement is issued when a follower is at point A, below:

GEAR follower **AT** target.ratio **IN** some.distance
BY

Forward pacer motion gears the follower into the segment AB.
Reverse pacer motion, before the GEAR AT is complete at point B, would gear the follower back through the profile, segment BA.
Continued reverse pacer motion would be EBCed, segment AA'.
At point B, **DSP.DONE@** would be **TRUE**, and the GEAR AT complete. Now with pacer forward and reverse motion, target.ratio is maintained, BC & BB'.



GEAR.RATIO@ shows current ratio

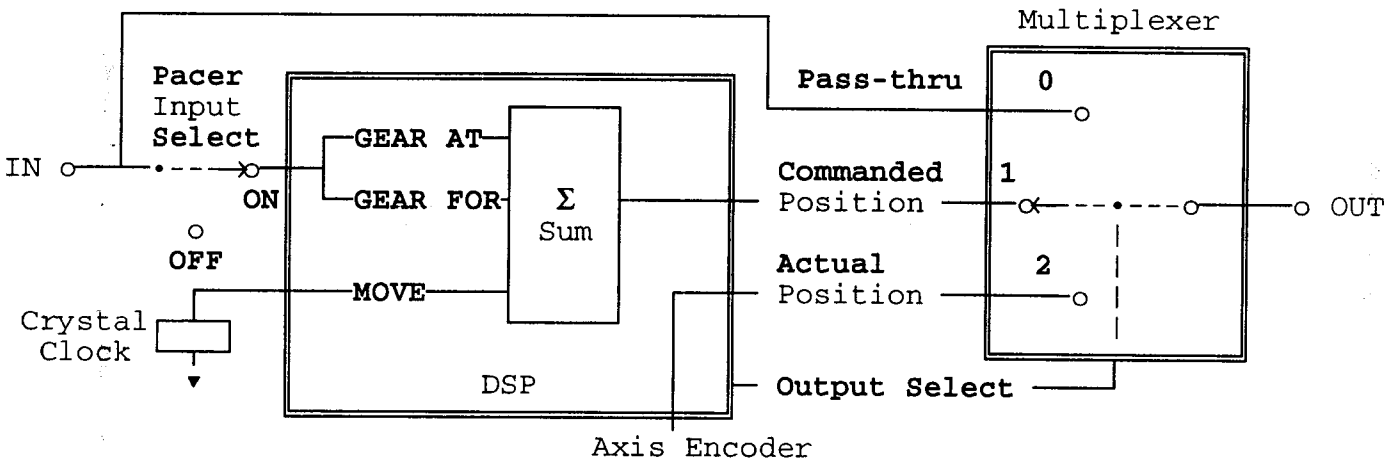
Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

When MotionDATA input is transferred directly by a **DSP Axis Module** from its input to its output by the **hardware multiplexer** there is no DSP computation overhead and essentially no delay in the information being transferred. This allows a large number of **follower** axes to receive simultaneous MotionDATA **pacer** information.

For example, if "Axis 1" is configured to transmit **pacer** information from the **actual position** of its **axis encoder**, and all the other axes in the system have their **MotionDATA Operating Mode** variable set to "pass-thru", then the actual position of Axis 1 is propagated to all other axes in the system simultaneously. They may all then operate as **follower** axes of Axis 1 using the **GEAR** statement.

Since any axis in the system can be reconfigured to provide real-time **pacer** information from its own commanded or actual position, this architecture allows **cascaded control** with a single position update time delay between cascaded sections of the MotionDATA communications link.

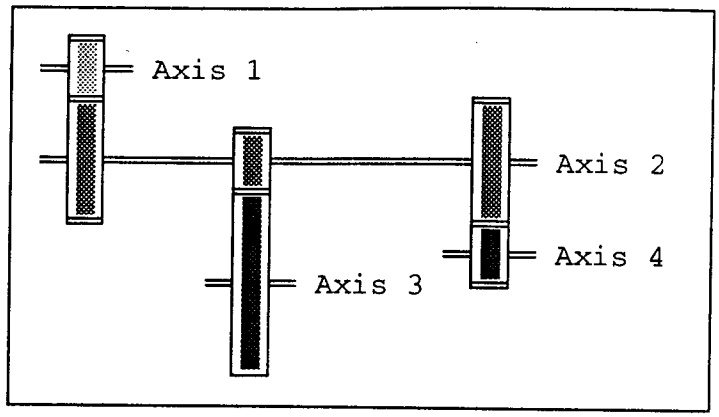
MotionDATA Interface



The second axis on a **Dual DSP Axis Module** can monitor MotionDATA only if **MD.MODE@** for the first axis is 10, 11, or 12 (monitoring is ON).

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Cascaded Control



The figure at the right illustrates cascaded control of an electronically **geared** system.

In this case, Axis 1 could be attached to a servo axis, or simply an **axis encoder**, as long as it is configured to generate **MotionDATA pacer** information.

Axis 2 can operate as a **follower** of Axis 1, and also generate MotionDATA on the basis of either its **commanded position** its **actual position.** Axis 3 & 4 can then operate as **follower** axes using the pacer information provided by Axis 2. The term *cascaded control* is used because follower motions of Axis 3 and 4 depend on the motion of Axis 2, which in turn depends on the motion of Axis 1.

Overtravel Limits

Generation III provides the safety feature of **Hardware Overtravel Limits** and **Software Overtravel Limits.** The intent of these features is to cease motion when one of these limits is encountered during the operation of a machine axis. There is a Forward and Reverse **Software Limit** available for each axis in the system. There may be a Forward and Reverse **Hardware Limit** available for each axis in the system.

This limit checking is only performed when an axis is in a torque-producing **mode.** Then an occurrence of **any** travel limit is latched in MotionBASIC.

When a servo's Hardware Overtravel Limit input is asserted or its position violates a Software Overtravel Limit, then:

- Commanded motion of the axis is stopped using **USR.ACL.LIM@,**
- **OTL.FWD@** or **OTL.REV@** Boolean variable for that axis is set **TRUE**
- A **Motion Error** will be generated which is **user trappable.**

See Also: **AXIS.OTL@**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Electronic Limit Switches

Generation III provides three programmable electronic limit switches for each **axis encoder** in the system. They turn ON and OFF automatically when an axis encoder's **actual position** goes "into" or "out of" a specified range.

Each electronic limit switch has a corresponding Boolean variable (**ELS1@ ELS2@** or **ELS3@**) as well as an inverted TTL-level output (74LS04) at the DSP I/O Header connector of the **DSP Axis Module**. Each TTL-level output is switched "ON" or "low" (less than 1 volt) when its corresponding Boolean variable is **TRUE**, and "OFF" or "high" (greater than 4 volts) when it is **FALSE**.

Electronic limit switches can be used as gating signals for **ASEN@** and **BSEN@**. This is done by selecting the proper **SENS.MODE@** for an axis.

Electronic limit switches may be used any time, but are particularly useful in conjunction with the **modulo position** feature.

Modulo Position

The modulo position feature allows the configuration of any axis so that its position will automatically "repeat" over a specified distance cycle. This is especially useful in rotating machines where you would like "position" for one or more of the axes to indicate the output position of the rotating tooling. e.g. 0-359 degrees

Both **commanded position** and **actual position** are affected, with the cycle distance specified by the **position modulus** variable. Enhancing this feature are three programmable **electronic limit switches** per axis, which turn ON and OFF automatically when the axis encoder's **actual position** is goes "into" or "out of" a specified range.

To setup a modulo position for an axis, set **POS.MOD@** for that axis to a non-zero positive number. Setting POS.MOD@ to zero disables modulo position reporting for that axis.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Pamux

Pamux is a parallel multiplexed I/O sub-system which can be used to provide a total of up to 512 discrete or analog I/O points. Pamux I/O points are added in multiples of 4, 8, or 16, with either a Digital or Analog "brain board" controlling each group. Because Pamux is a "parallel multiplexed" I/O sub-system, I/O access times are virtually the same as local discrete I/O points.

Pamux utilizes an eight line parallel bus, and supports 32 groups of up to sixteen I/O. "Brain boards" are available which provide either analog or digital I/O and their addresses are set with jumpers as described in the installation instructions. Both analog (**AIO@**) and digital (**DIO@**) I/O provided by Pamux are optically isolated.

To use Pamux I/O, the **EIO.MODE@** variable must be set to configure the **Extended I/O Connector** as "Pamux compatible". The **IO.MODE@** variable is then used to configure the individual Pamux I/O points consistent with the "brain boards", their addresses, and the installed analog and/or digital I/O modules.

MotionPRO

MotionPRO software provides a complete Motion Programming environment on any IBM-PC or true compatible. It provides user-friendly "pull-down menus" and "pop-up windows" and includes the following capabilities:

- This hypertext MotionBASIC reference manual on-line at all times;
- Interactive serial communications to the motion controller:
  - extensive interactive program debug tools,
  - automatic screen-oriented configuration (See **MP.CONFIG**),
  - context sensitive help
  - **MotionBASIC File** Management
  - **MotionPRO directives**
- A **text editor** with pull-down menus and file management system.
- An extensive library of MotionBASIC programming tools

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Example: The following example establishes events. One event will be Discrete Input #7 and the other will be an MMI Function Key, F2.

```
  INIT:
  IO.MODE@(7)="R"                        'Event Inputs should be latched
  ON EVENT DIO@(7) GOSUB IO.HANDLER      'Definition Step for Input #7
  ON KEY(2) GOSUB KEY.HANDLER            'Definition Step for Function Key F2
  OPEN "R",0,"SRL2:MMI"                  'Opening the MMI device
  MAIN:
  EVENT DIO@(7) ON                       'Enable the trapping of Input #7
  KEY (2) ON                             'Enable the trapping of Function Key F2
  EVENT ON                               'Enable Event Processing
* WHILE TRUE                             'Start of an endless loop
*   PRINT @ 4,4;USING;"###";J;           'Print a count to the MMI screen
*   J=J+1                                'increment the count
*   WAIT 250                             'delay 1/4 sec
* WEND                 * Represents statements where trapping is ON,
  END                    and the event routines could be serviced.
```

When event routines (& error handler routines) are called, implicit commands are issued by MotionBASIC to prevent another event from interrupting these routines while they are processing.

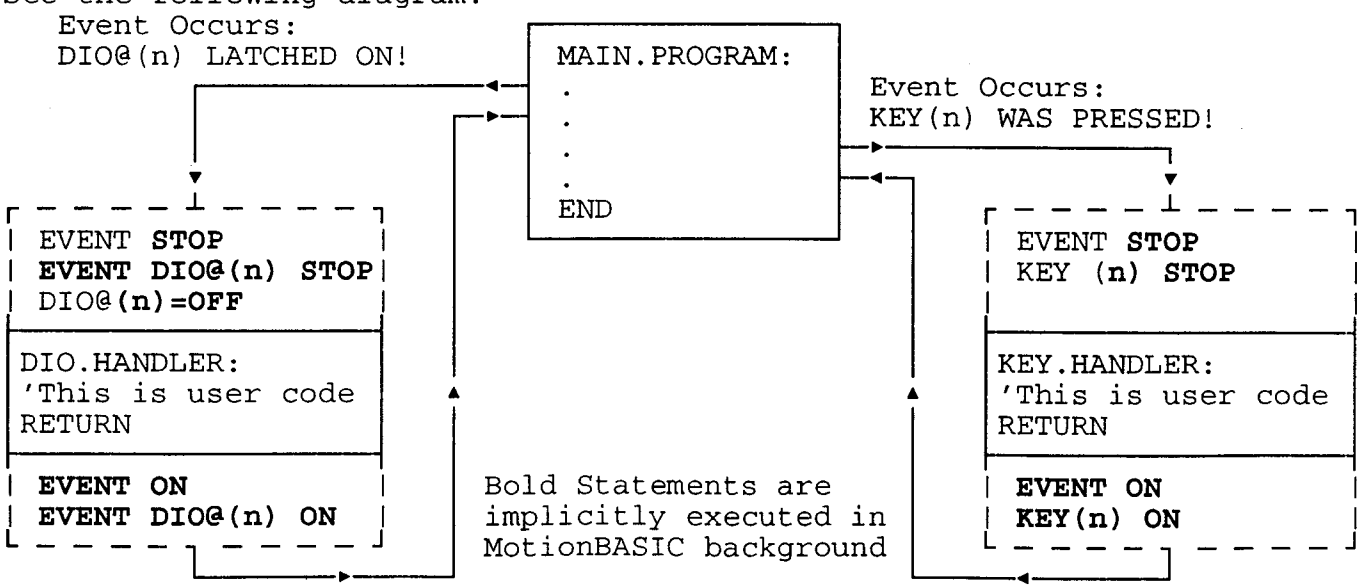
The implicit actions taken by MotionBASIC are:

Global Event Trapping is placed in the STOPPED state. In this way events that occur during the execution of an event handler are latched, but not serviced during the execution of an event handler routine.

The event being serviced is placed into the STOPPED trapping state. This is done so a user can easily enable global trapping of "all other events" within the event handler, but the event being serviced will not be able to interrupt itself.

Global Event Trapping and the serviced event's trapping state are reset to ON after a **RETURN** is executed at the end of the handler routine.

See the following diagram:



Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionBASIC Program Files

Program files typically contain **statements**, **functions**, **directives**, **comments** & **disk comments**. They can be stored on **disk** & **Memory Disks**.

Program files should have the filename extension, "**.BAS**". Files with a BAS extension are handled in a special manner during a transfer from **disk** to **Memory Disk**. MotionPRO **Directives** and line numbering tasks are obeyed during a .BAS file transfer. **Files with .BAS extensions are stored on Memory Disk with line numbers, but without disk comments.**

To **RUN** a program, the program first needs to be "loaded" into the motion controller. The source for a **LOAD** could be a **disk** or on **Memory Disk**. During the load, the source program code is tokenized and placed into controller **non-volatile memory**. After the load the program can be **RUN**.

The commands and statements that are specifically used with program files are: **LOAD** **SAVE** **RUN** **MERGE** **CHAIN**

See Also: **Data Files** **Program Encryption**

MotionBASIC Program Encryption

MotionBASIC provides the user with the ability to "encrypt" program code. The purpose of encrypting program code is to protect one's intellectual property. There are two levels of encryption protection:

1) The user program cannot be altered, viewing IS possible (from Gen III).
2) The user program cannot be altered, viewing IS NOT possible.

The **SAVE** command options allow the user to encrypt code.

WARNING!!! DO NOT SAVE ENCRYPTION ONTO YOUR ORIGINAL SOURCE FILE NAME!! YOU WILL LOSE YOUR WORK AND NOT BE ABLE TO RECOVER IT!!

A program that has been **LOAD**ed into a controller, may be **SAVE**ed to a file. The destination of this save process is a file either on a **disk** of the development computer, or on a **Memory Disk** plugged into the controller.

IN A NORMAL **SAVE** PROCESS, the result is an ASCII text file which CAN be read & edited with a text editor. The default file extension is **.BAS**.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionBASIC Sequential Files

A Sequential File is one of the two types of MotionBASIC **Data Files.** In a sequential file, data is written as a series of **ASCII** characters. Data is stored, one item after another (sequentially), in the order sent. Data is read back in the same way.

Sequential files are easier to create than are **Random Access Files,** but are somewhat limited in flexibility and speed when accessing data. A Sequential file will require more room on the **Memory Disk** than would a **Random Access File** which is stored in a compressed format.

The following statements and functions are used with sequential files:

OPEN File	**PRINT #**	**INPUT #**	**LOC**	**EOF**
CLOSE	**PRINT # USING**		**LOF**	

See Also: **Creating Sequential Files**
Accessing Sequential Files
Adding Data to Sequential Files

Creating Sequential Files

The following program steps are required to create a sequential file:

1) **Open file** in Output mode. This will create the file if it does not exist, or will destroy its current contents if it does exist.

   example: OPEN "O",#1,"1:\REPORT\BATCH.REP"

2) Write string data to the file using the **PRINT #** statement. Example:

```
A=1 : B=2 : C=3 : D=4
PRINT #1, USING "###.##,";A,B,C;
PRINT #1, USING "###.##" ;D
CLOSE 1
```

'the resulting data file

```
 1.00,  2.00,  3.00,  4.00<cr>
EOF
```

3) **CLOSE** the file.

Note: **<cr>** is carriage return

See Also: **LOC** **Accessing Sequential Files** **Adding Data to Sequential Files**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionBASIC Random Access Files

A Random Access File is one of the two types of MotionBASIC **Data Files**. In a random access file, data is stored and retrieved in distinct, numbered units called **records**.

Since records are referenced by number, specific data can be read from a record location; the program need not read the entire file, as it must with a **Sequential File**.

Creating and accessing random access files requires more program steps than sequential files, but they consume less **Memory Disk** space.

The following statements and functions are used with random access files:

OPEN File	**LSET**	**GET**	**EOF**
FIELD	**RSET**		**LOC**
CLOSE	**PUT**		**LOF**

See Also: **Creating Random Access Files** **Accessing Random Access Files**

Creating Random Access Files

The following program steps are required to create a random access file:

1) **Open file** in random access mode. This will create the file if it does not exist. The example specifies a record length of 32 bytes.

   example: OPEN "R",#1,"1:\REPORT\BATCH.REP",32

2) Use the **FIELD** statement to allocate space in the random buffer.

3) Use **LSET** or **RSET** to move data into the random buffer fields.

4) Write the data from the random buffer to the **Memory Disk** using the **PUT** statement.

The **LOC** function when used with a random access file, returns the current record number.

See Also: **Accessing Random Access Files**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionPRO Text Editor

The text editor provided with **MotionPRO** is integrated through the pull-down menu structure and provides powerful yet easy-to-use editing of MotionBASIC programs stored on **disk**, as well as other text files. It is menu-driven, provides context sensitive on-line help and has special macros provided for editing MotionBASIC programs. In conjunction with the **MotionPRO directives**, it allows the user to edit MotionBASIC programs without line numbers, and include programs in multiple files.

It is not to be confused with the **line editor**, which is used to edit MotionBASIC programs in the non-volatile memory of the Motion Controller.

Disk

The term **disk** refers to the disk drive(s) of an IBM-PC or compatible computer running the **MotionPRO** development software.

This computer running **MotionPRO**, is also called the Development System. It is connected to a Generation III controller through a RS-232 serial communications link. The RS-232 port is called the "Development Port" on the motion controller.

The disk drive(s) of the Development System will typically have letters identifying its drives: (e.g. **C:** & **D:** are typically Hard Drive designators while **A:** & **B:** are typically diskette drive designators).

MotionBASIC Files can be stored in any of these disk locations through file management utilities and commands executed at this system.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Syntax & Usage Notation Table

Symbols	Description	Meaning
[ ]	Square Brackets	Optional clause or parameter
{ | }	"Curly" Braces	Choices are available
...	Three Dots	Preceding clause is repeatable

This is a "quick-reference" of syntax notation symbols. For more detailed descriptions and examples, see: **MotionBASIC Syntax Notation**

See Also: **Typical Usage**

MotionBASIC Syntax Notation

MotionBASIC expects **commands**, **statements**, and **functions** to conform to a specific structure, or **syntax**. In this manual, each **keyword** is presented with syntax notation to indicate just how a statement using that keyword can be properly written. If a legal syntax is not used, a **Syntax Error** is generated by MotionBASIC.

[] SQUARE BRACKETS **[optional]** In syntax notation, items within square brackets are optional. A legal statement can be composed with optional items completely omitted. If the optional clause is used, the brackets should not be typed. The brackets are not part of the statement, but merely syntax notation marks.

{ | } BRACES **{ choice1 | choice2 }** In syntax notation, "curly" braces indicate a choice is available. The choices, inside the braces, are separated by the | character. The braces should not be typed. The braces are not part of the statement, but merely syntax notation marks.

... ELLIPSES **[,repeatable] ...** In syntax notation this indicates

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionBASIC Commands

NAME	Change the name of a Memory Disk file
NEW	Delete the program currently in non-volatile memory
NEW RESTORE	Restore non-volatile memory
RENUM	Renumber all or part of a program
RMDIR	Delete a directory on the Memory Disk
RUN	Execute the current program
SAVE	Write the current program to a **program file**

MotionBASIC Statements

Statements are executable instructions in MotionBASIC which cause it to perform a desired action such as "perform a computation" or "move a servomotor". Statements can be executed in **direct mode**, but they are primarily put together in a logical sequence to create programs to be executed later in **indirect mode**. MotionBASIC statements fall into one of the categories below:

Motion Statements
Control Structures
Display/Keyboard Statements
External I/O Statements
Program Initialization Statements
Miscellaneous Statements
Line Format for Statements

All Statements (alphabetical list)

Motion Statements

MOVE The **MOVE statement** can be used to:
- **move for** a relative distance from the current position,
- **move to** an absolute position, or
- **move at** a specified speed.

GEAR The **GEAR statement** can be used to:
- **gear for** a relative distance from the current position,
- **gear at** a specified ratio to the pacer encoder.

HALT The **HALT statement** can be used to:
- stop motion on the designated motors.

Motion statements transfer actual command information to the **DSP Axis Modules** through the **Motion Queue.**

USE EXTREME CAUTION WHEN INTERACTIVELY EXECUTING MOTION STATEMENTS, especially if ACL.MAX@, DCL.MAX@ **or** SPD.MAX@ **are set to high values!**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Display/Keyboard Statements

Display Statements:

PRINT	Display on screen
CLS	Clear screen
CLREOL	Clear to end of line
COLOR	Set foreground, background colors
LOCATE	Position the cursor
WIDTH	Sets the output width

Keyboard Statements:

INPUT	Get input from keyboard
INPUT @	Get input from keyboard at specified location
KEY	Set or display soft keys
KEY (n)	Enable/disable key trapping
ON KEY	Trap for keypress

See also: **Input/Output Functions**

External I/O Statements

Character Device and File I/O Statements:

OPEN	Allow device or file I/O
CLOSE	Close a device or file
PRINT #	Output to a device or file
INPUT #	Get input from a device or file
WIDTH	Set output width
TALK	Console talk mode
CHAIN	Load and execute a program or overlay
FIELD	Define a Random Access File field
GET	Read from a Random Access File
PUT	Write to a Random Access File
IOCTL	Send device "control data" string

See also: **Input/Output Functions**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

CHAIN	Load and execute program or overlay
CLEAR	Reset variables / set maximum string space
CLOSE	Close a device
CLREOL	Clear to end of line
CLS	Clear screen
COLOR	Set foreground, background, and border colors
DATA	Store constants for retrieval via **READ**
DEF FN	Define function
DEF SEG	Define segment
DEFDBL	Define variable(s) as double precision
DEFINT	Define variable(s) as integer
DEFLNG	Define variable(s) as long integer
DEFSET	Define variable(s) as set
DEFSNG	Define variable(s) as single precision
DEFSTR	Define variable(s) as string
DIM	Dimension array(s)
END	Terminate program
ERASE	Remove array from memory
ERROR	Force a MotionBASIC error
EVENT	Enable/Disable Hardware Event Trapping
FIELD	Define **Random Access File** field format
FOR	Begin definition of FOR / NEXT loop
GEAR	Gear one or more servo motors to the electronic lineshaft
GET	Read from a random access file
GOSUB	Execute subroutine
GOTO	Unconditional branch
HALT	Halt a motion in progress
IF	Conditional branch
INPUT	Get input from keyboard
INPUT #	Get input from a device
INPUT @	Get input from keyboard at specified location
IOCTL	Send control data to a device driver
KEY	Set or display soft keys
KEY (n)	Enable/disable key trapping
LET	Assignment
LOCATE	Position the cursor
LSET	Left-justify string data
MOVE	Move one or more servomotors
NEXT	End definition of **FOR**/NEXT loop
ON ERROR	Enable/Disable error trapping
ON EVENT	Trap for specified Event
ON KEY	Trap for keypress
ON...GOSUB	Branch to item in subroutine list
ON...GOTO	Branch to item in line number list
OPEN	Allow device I/O
OUT	Send byte to output port
OUTW	Send word to output port
POKE	Write byte to memory
POKEW	Write word to memory
PRINT	Display on screen
PRINT #	Output to a device
PUT	Write to a **random access file**
RANDOMIZE	Reseed random number generator
READ	Assign **DATA** items to variables

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

String Functions

INSTR	Find substring
LCASE$	Convert upper cases to lower cases
LEFT$	Substring at left
LEN	Length of string
MID$	Substring in middle
RIGHT$	Substring at right
SPACE$	String of N spaces
STRING$	String of specified length and character
UCASE$	Convert lower cases to upper cases

See Also: **Garbage Collection**

Conversion Functions

ASC	**ASCII** value of first character
BIN$	Binary value, as string
CHR$	Convert ASCII code to character
HEX$	Hexadecimal value, as string
OCT$	Octal value as string
STR$	String representation of numeric expression
VAL	Numeric value of string

Input/Output Functions

Display Functions:

CSRLIN	Row position of cursor
POS	Current cursor column position
SPC	Skip N spaces in a **PRINT** statement
TAB	Tab to specified position in a **PRINT** statement

Keyboard Functions:

INKEY$	Most recent character at keyboard

Character Device I/O Functions:

EOF	End-of-file status
INPUT$	Read specified number of characters
SPC	Skip N spaces in a **PRINT #** statement
TAB	Tab to specified position in a **PRINT #** statement
LOC	Last accessed record
LOF	Length of file

See also: **Display/Keyboard Statements** **External I/O Statements**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionBASIC Functions

INKEY$	Most recent character at keyboard
INP	Read from an I/O port
INPUT$	Read specified number of characters
INPW	Read from an I/O port
INSTR	Find substring
INT	Next lower integer
LCASE$	Convert upper cases to lower cases
LEFT$	Substring at left
LEN	Length of string
LOC	Last accessed record
LOF	Length of file
LOG	Natural logarithm
MBVER$	MotionBASIC version
MID$	Substring in middle
OCT$	Octal value as string
PEEK	Value at specified address
PEEKW	Value at specified address
POS	Current cursor column position
RIGHT$	Substring at right
RND	Random number
SGN	Sign of number
SIN	Sine
SPACE$	String of N spaces
SPC	Skip N spaces in a **PRINT** statement
SQR	Square root
STR$	String representation of numeric expression
STRING$	String of specified length and character
TAB	Tab to specified position
TAN	Tangent
TIME$	Set system time or return it as a string
TIMER	Return elapsed time in **user units**
UCASE$	Convert lower cases to upper cases
VAL	Numeric value of string

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Precedence **Operator Precedence in MotionBASIC**

Highest:	- (unary negation)
	^ (exponentiation)
	* / \ **MOD**
	+ -
	= <> < > <= >= **WITHIN**
	NOT
	AND
Lowest:	**OR XOR**

Operators in MotionBASIC have an inherent precedence. In the absence of explicit grouping (with parentheses), an expression is evaluated in the order dictated by this inherent precedence. Operators of the same precedence level are evaluated in left-to-right order.

All function calls in an expression are evaluated before any operators.

See also: **Differences**

Bitwise Operations

The four logical operators **NOT**, **AND**, **OR**, **XOR** perform "bitwise" operations on integer expressions. The term bitwise means that the operator performs its operation on each bit (or pair of bits) of the integer (or integers) being operated upon. The **truth tables** for these operators show the result for each bit after the operation is complete.

Example:

```
10 LOGICAL1% = &B0000000011111111 : LOGICAL2% = &B1100110011001100
30 RES.NOT% = NOT LOGICAL1% : RES.AND% = LOGICAL1% AND LOGICAL2%
50 RES.OR% = LOGICAL1% OR LOGICAL2% : RES.XOR% = LOGICAL1% XOR LOGICAL2%
70 PRINT BIN$(RES.NOT%) ;BIN$(RES.AND%) ;BIN$(RES.OR%) ;BIN$(RES.XOR%)
80 PRINT NOT {2,10,11,15,19,28,29,30,31}
90 END
RUN
1111111100000000 11001100 1100110011111111 1100110000110011
1,3,4,5,6,7,8,9,12,13,14,16,17,18,20,21,22,23,24,25,26,27,32
```

Note the leading zero suppression when RES.AND% is printed.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionBASIC Constants

Examples:	1234, &O0033, &Hab, &B0111100	type **integer**
	1234&, 123456, &H123&, -23&	type **long integer**
	1234#, 1.234, -3456. 2.3e35	type **double**
	{}, {1,15},{14,17,32}	type **set**
	"This is a string"	type **string**

MotionBASIC Variables and Data Types

Variables are names used to represent values used in a MotionBASIC program, and they may be any of the *data types* listed below. The value of a variable may be assigned explicitly by the programmer, or it may be assigned as the result of calculations in the program. Non-initialized numeric variables have a value of zero.

Variable Names **Memory Usage** **Constants** **Boolean** **Set**
Array Variables **Type Conversion** **Differences** from GWBASIC

Data type	Suffix	Range	Accuracy	Storage
Integer(default)	%	-32768 to 32767	Perfect	2 bytes
Long Integer	&	-2147483648 to 2147483647	Perfect	4 bytes
Single Precision	!	± 3.4E-38 to 3.4E38	6 digits	4 bytes
Double Precision	#	± 1.07E-308# to 1.07E308#	16 digits	8 bytes
Set	~	0 to 32 elements	Perfect	4 bytes
String	$	0 to 255 characters		5+bytes
Empty **String**	$	0 characters		2 bytes
ORMEC Variables	@			

Variable Names and Type Declaration Characters

MotionBASIC variable names may be any length and all characters are significant. Variable names can contain letters, numbers, and the decimal point. However, the first character must be a letter. Special type declaration characters are also allowed (see below).

A variable name may not be a reserved word, but embedded reserved words are allowed. **Reserved words** include all MotionBASIC commands, statements, function names, and operator names. If a variable begins with **FN**, it is assumed to be a user-defined function. Variables may represent either a numeric value or a string.

String variable names are written with a dollar sign ($) as the last character. For example:

```
A$ = "SALES REPORT"
```

The dollar sign is a variable type declaration character; that is, it "declares" that the variable will represent a string.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Boolean variables are actually **integer** variables which have only two states: TRUE (ON) and FALSE (OFF). They are declared just like any other integer variable.

TRUE is represented by the value -1, and FALSE by 0, because the values -1 and 0 represent integers with all bits set to one, or cleared to zero respectively. MotionBASIC provides special keywords, **TRUE**, **FALSE**, **ON** and **OFF** to represent Boolean constants. The two Boolean values are equivalent to the numeric constants below:

Base	TRUE, ON	FALSE, OFF
Decimal	-1	0
Hexadecimal	&HFFFF	&H0000
Binary	&B1111111111111111	&B0000000000000000
Octal	&O177777	&O000000

Since the values -1 and 0 represent integers with either all bits set to one or cleared to zero, only these values ensure the proper operation of the **bitwise** logical operators **AND**, **OR**, **XOR**, and **NOT**, with Boolean expressions.

The boolean value of a **set** is FALSE if the set is empty, otherwise, it is TRUE.

Example:

```
10 RESULT% = TRUE OR FALSE
20 PRINT TRUE; FALSE; RESULT%; ON; OFF
30 END

RUN
-1  0  -1  -1  0
```


Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

4. When expressions are evaluated, all operands are normally converted to the same level of precision--that of the highest-precision operand in the expression. For expressions which require **integer** or **long integer** operands, **integer type conversion** is performed.

Note: When an integer variable is assigned a floating point value, the value is rounded to the nearest whole number. If the fractional part of the floating-point number is exactly 0.5, the value is rounded to the nearest even integer.

```
Example 1      Example 2      Example 3      Example 4
 A%=2.5         A%=2.51        A%=3.5         A%=3.49
 PRINT A%       PRINT A%       PRINT A%       PRINT A%
 2              3              4              3
```

5. Sets may only be assigned to a **set**, **integer**, or **long integer** variable. If a set is assigned to an integer, or long integer variable, the variable takes the value of the set's smallest element.

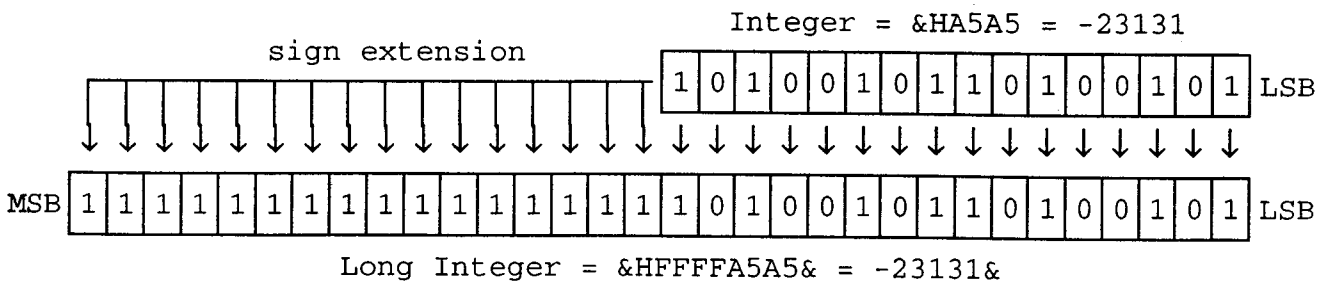
6. If an integer or long integer is to be assigned to a **set** variable, the value is treated as an element for the set and thus the value must be between 0 and 32. Assigning an integer value of 0 to a set variable results in the "empty" set. i.e. the set with no elements, as designated by "{}"

7. A "type mismatch" error will be generated whenever a floating point value is assigned to a set variable or when a set value is assigned to a floating point variable.

Integer Type Conversion

Integers are converted to long integers by a process called "sign extension". Sign extension preserves the value of **two's complement** negative numbers through the conversion.

The 16-bit integer value is copied directly into the low 16 bits of the long integer. The sign bit (most significant bit) of the two's complement integer is copied into the 16 most significant bits of the long integer.



Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Set

A *set* is a collection of elements. Each element is represented by a bit in memory. Since each *set* in MotionBASIC takes 4 bytes of storage, a set can have up to 32 elements, numbered from 1 to 32.

Set constants are defined by including a *set value* string within "braces". e.g. The expression **{1,2,3}** defines a set with three elements. Only constants between 1 and 32 are allowed to be elements within the "braces". Set variables are defaulted to the "empty" set as designated by **{}**.

Sets are most commonly used in MotionBASIC to define groups of servo axes. These groups can then receive a common command or have a common value assigned to any pre-defined variable indexed by **axes**. The pre-defined variable **AXIS.LIST@** defines the set of all servos in the system. **AXIS.SET@** defines a default group of servos for use in MotionBASIC statements indexed on ***axes***. **AXIS.FAULT@** defines a group of servos which currently have a "fault". **AXIS.OTL@** defines a group of servos which currently have an over travel limit.

Example:

```
10 PRINT AXIS.LIST@
20 AXIS.SET@ = {1}
30 PRINT AXIS.SET@
40 END
RUN
{1,2,3,4}
{1}
```

See also: **Type Conversion** **VAL** **STR$** Example under **AXIS.SET@**
Impact of Set Variables on MotionBASIC

Integer

Integer variables are 16-bit binary values which may contain any whole number from -32768 to 32767. Integer variable name may end with a percent sign (%). Storage space for integer values is 2 bytes (16-bits).

Because it is designed for real-time control, MotionBASIC defaults all variables to integer unless declared otherwise!

Integer constants can denote several different radixes. They may be written in one of the following forms:

Decimal: [-]*ddddd*[%] Hexadecimal: [-]&H*dddd*[%]
Octal: [-]&[O]*dddddd*[%] Binary: [-]&B*dddddddddddddddd*[%]

where *d* represents a digit in the appropriate radix.

Any integral constant followed by a percent sign (%) is considered to be an integer, as is any integral constant with a value small enough to be an integer.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Single Precision

Single Precision floating-point variables may contain any positive or negative number from 3.4E-38 to 3.4E+38. Up to six significant digits may be included in the mantissa before rounding will occur. They may be explicitly declared as single precision with the **DEFSNG** statement, or by ending the variable name with a exclamation point (!). Storage space for single precision values is 4 bytes (32-bits).

Single Precision floating-point constants may be written in one of the following forms:

[-]*ddd.ddd*[E[-]*ee*]	[-]*ddd.ddd*[E[+]*ee*]
[-]*ddd*[*.ddd*][E[-]*ee*]!	[-]*ddd*[*.ddd*][E[+]*ee*]!

where *ddd.ddd* represents the mantissa with up to six significant decimal digits (the decimal point may be located anywhere in the mantissa), and *ee* represents the exponent with one or two digits.

All floating-point constants are internally represented as **double precision**. There is no way to explicitly specify a **single precision** constant. Any decimal number followed by an exclamation point (!) or a number sign (#) is considered to be a double precision floating-point number.

Since all floating-point arithmetic is performed in double precision, and all floating-point expressions used as function arguments are promoted to double precision, it might appear you would have little use for single precision. Single precision variables however, require only four storage bytes, whereas double precision requires eight. The decision between single and double is one of space versus run-time conversion cost.

The IEEE representation consisting of a sign bit, a 7-bit excess 127 binary exponent, and a 24-bit mantissa is used. The mantissa represents a number between 1.0 and 2.0; since the high-order bit of the mantissa is always 1, it is not stored in the number.

Examples:
```
VALUE! = 1.234
F1! = 1.234E-24
VALUE = 123!
```

See also: **Double Precision** **Long Integer**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

String Variables

String variables contain variable length groups of **ASCII** characters. These character strings may be from 0 to 255 characters in length, and may contain any combination of characters. If not declared with the **DEFSTR** statement, a string variable name must end with a dollar sign ($). The size of a string variable is 5 bytes for the descriptor plus one byte per character in the string. Non-initialized string variables contain an "empty" or "null" string, which is a string with no ASCII characters in it. **String variables are volatile and will be reset to the "null" string on** powerup.

String constants may be written as a string of characters enclosed in double quotes, e.g. "*string*", where *string* represents a string of between 0 and 255 **ASCII** characters.

Example: VALUE$ = "abcd"

See also: **Garbage Collection**

String Garbage Collection

The speed at which a MotionBASIC program executes is normally very predictable. When using strings, however, the programmer should be aware of a process called "string garbage collection", which can sometimes cause a string operation to take longer to execute. e.g. String garbage collection for a 4,000 byte string space would typically take 10 to 20 msec. For the default string space size, 1024 bytes, 9.6 ms worst case.

Since garbage collection occurs only during string operations, and only when insufficient string space exists, following the rules below will prevent unexpected delays in a program:

1. Do not perform string operations in critical (fast) code sections.
2. To reduce the time needed for each string garbage collection, reduce the number of string variables used in the program (i.e. no large string arrays).
3. To reduce the frequency of string garbage collections, increase the amount of available string space by using the **CLEAR** statement.
4. Use the **FRE** function to force garbage collection in non-critical sections of code.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Unit Parameters and I/O Variables

Unit I/O Setup Parameters:

IO.MODE@	Discrete I/O Config	**EIO.MODE@**	Extended I/O Config

Unit I/O Variables

DIO@	Discrete I/O Point	**AIO@**	Analog I/O Point

Unit Status Parameters:

LOOP.RATE@	Servo Loop Update Rate	**AXIS.LIST@**	**Set** of all axes
AXIS.SET@	Default **set** of axes	**UNIT.ID@**	Unit ID Code
FAULT@	Unit Fault Status	**AXIS.FLT1@**	First axis to fault
AXIS.FAULT@	All axes with a fault	**PG.LEN@**	Page Length
AXIS.OTL@	All axes with an OTL	**BAUD@**	Serial Port Baud Rate
PW@	Program Write Enable	**PB1@**	Pushbutton PB1 pressed
ESTOP.OK@	Emergency Stop Input OK	**USER2.LED@**	User 2 LED ON
USER1.LED@	User 1 LED ON		

User Units

MotionBASIC provides built-in conversion factors so that time, position, speed and acceleration can be expressed in units of the user's choice. e.g. msec, inches, inches/sec, & g's. Sections on **axis position units**, **speed units**, **acceleration units**, **pacer position units** and **gear ratio multipliers** describe the conversion factors below:

CNT.REV@		Axis Position Transducer counts/rev
MTR.SPD.LIM@	**USR.SPD.LIM@**	Motor/Machine Speed Limits
MTR.ACL.LIM@	**USR.ACL.LIM@**	Motor/Machine Acceleration Limits
PCT.REV@	**PCR.SPD.LIM@**	Pacer-counts/rev and Speed Limit

After setting LOOP.RATE@, **set the parameters above in the order listed. All** servo loop tuning parameters **must then be initialized.**

POS.MUL@	**PPS.MUL@**	Axis/Pacer Position Units Multiplier
POS.DIV@	**PPS.DIV@**	Axis/Pacer Position Units Divider
OUTSPD.MUL@	**INSPD.MUL@**	Gear Ratio Output/Input Speed Multipliers
TIME.MUL@		Time Conversion Factor

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

User Speed Units

MotionBASIC automatically calculates the conversion factor for user speed units when the user establishes the "absolute machine speed limit" both in terms of "user speed units" (**USR.SPD.LIM@**) and the corresponding "absolute motor speed limit" (**MTR.SPD.LIM@**) in RPM of the axis encoder.

Example 1: You want speed and velocity units to be expressed in the MotionBASIC program in RPM, and the fastest you ever want the machine to go is 4000 RPM. The motor is coupled directly to the machine and so it also goes 4000 RPM. Set **USR.SPD.LIM@** = 4000 & **MTR.SPD.LIM@** = 4000. These values are the defaults.

Example 2: You want speed and velocity units to be expressed in the MotionBASIC program in inches/min, and the fastest you ever want the machine to go is 500 in/min. The motor is driving a 5 pitch ballscrew (5 revs/inch), and so it runs at a corresponding speed of 2500 RPM. Set **USR.SPD.LIM@** = 500 & **MTR.SPD.LIM@** = 2500.

Note that the process of setting the "user speed units conversion factor" utilizes the **CNT.REV@** parameter because the internal units for speed are "counts per second" of the position transducer. This process also determines the speed resolution of the system and affects the servo loop tuning parameters. User speed units are generally set only once during program initialization. There is another parameter, **SPD.MAX@**, for changing the maximum speed during the applications program.

User Acceleration Units

MotionBASIC automatically calculates the conversion factor for user acceleration units when the user establishes the "absolute maximum machine acceleration limit" both in terms of "user acceleration units" (**USR.ACL.LIM@**) and the corresponding "motor acceleration rate" (**MTR.ACL.LIM@**) in rev/sec^2 of the axis encoder.

Example 1: You want acceleration units to be expressed in the MotionBASIC program in rad/sec^2, and the fastest you ever want the machine to accelerate is 20,000 rad/sec^2. The motor is coupled directly to the machine and so it also accelerates at 20,000 rad/sec^2, which is equivalently expressed as $20000/2\pi$ = 3183 revs/sec^2. Set **USR.ACL.LIM@** = 20000 & **MTR.ACL.LIM@** = 3183.

Example 2: You want user acceleration units to be in inches/second^2, and the maximum acceleration you want to allow is 500 in/sec^2. The motor is driving a 5 pitch ballscrew (5 revs/inch), and so the motor acceleration when the axis is at 500 in/sec^2 is 2500 revs/sec^2. Set USR.ACL.LIM@ = 500 & MTR.ACL.LIM@ = 2500.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Gear Ratio Multipliers

The MotionBASIC statement *GEAR AT 1 TO 1* specifies a user gear ratio. The user ratio "1 to 1" would have an application specific meaning. For actual control by the DSP these "user ratios" are converted to encoder count ratios; the "ratio of **axis encoder** counts to **pacer** encoder counts". To perform the conversion, MotionBASIC internally multiplies the **outspeed** and **inspeed** parameters of the **GEAR** statement by **OUTSPD.MUL@** and **INSPD.MUL@**, respectively.

Example 1: A **pacer** encoder has 24,000 cts/rev and the **follower** has 6000 cts/rev, then **GEAR AT 1 TO 1** without gear ratio multipliers would result in the **follower** moving 4 times faster than the **pacer**. To have "*1 TO 1*" result in the same speed, set **OUTSPD.MUL@** =1 & **INSPD.MUL@** =4. The follower then moves 1 count for every 4 pacer counts.

Example 2: With the encoders above, for user ratios that specify speed percentages, set **OUTSPD.MUL@** = 1 and **INSPD.MUL@** = 400. Then the statement *GEAR AT 50* will move follower to pacer encoders at 50 counts to 400 counts, (1 count for every 8 pacer counts), a 50% speed ratio.

When **GEAR AT 1 TO 1** is required to match machine rotary speeds, giving 1 cycle of the follower axis for 1 cycle of the pacer axis, then the gear speed multipliers are as follows:

$$\frac{\textbf{INSPD.MUL@}}{\textbf{OUTSPD.MUL@}} = \frac{\text{COUNTS for 1 Machine pacer axis cycle}}{\text{COUNTS for 1 Machine follower axis cycle}}$$

When the application requires GEAR AT 1 TO 1 to match machine linear speeds, meaning move X distance of the follower axis for X distance of the pacer axis, then the gear speed multipliers are as follows:

$$\frac{\textbf{INSPD.MUL@}}{\textbf{OUTSPD.MUL@}} = \frac{\textbf{PPS.MUL@} * \textbf{POS.DIV@}}{\textbf{PPS.DIV@} * \textbf{POS.MUL@}}$$ (all referenced to follower)

NOTE: Reduce these fractions to a numerator and denominator that are integers. **DO NOT** perform floating point divisions or rounding.

Axis Range Variables

SPD.MAX@	Maximum Speed
ACL.MAX@	Maximum Acceleration Rate
DCL.MAX@	Maximum Deceleration Rate
DRV.MAX@	Maximum Drive Command
STL.FWD@	Forward Software Travel Limit
STL.REV@	Reverse Software Travel Limit
PERR.MAX@	Maximum Position Error
PERR.INPOS@	In-Position Error

All Axis Range Variables are indexed by ***axes***.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Numeric Axis Status Parameters

Read-Write Parameters:

AFAULT@	Axis Fault Code
DRV.CMD@	Drive Command
VEL.CMD@	Velocity Command
POS.ACT@	Actual Position
DIST.BIAS@	Distance Bias
DSP.CTR@	DSP Motion Counter

Read-Only Parameters:

ACL.CMD@	Accel Command
VEL.ACT@	Actual Velocity
POS.CMD@	Position Command
DIST.MOVE@	MOVE distance
DIST.GEAR@	GEAR distance
DIST.GRAT@	GEAR AT distance
POS.ERR@	Position Error
PERR.NORM@	Normalization Error

Read-Only Parameters, cont'd:

AIN1@	Axis Analog Input One
AIN2@	Axis Analog Input Two
ALARM@	Servodrive Alarm Code
GEAR.RATIO@	Current Gear Ratio
POS.ABS@	Absolute Encoder Poll
POS.ASEN@	Position at A-Sensor
POS.BSEN@	Position at B-Sensor
POS.DELAY@	Position at Delay Input
POS.ZREF@	Position at Encoder Ref
ACL.LAST@	Last Specified Accel
DCL.LAST@	Last Specified Decel
VEL.LAST@	Last Specified Top Vel

All Numeric Axis Status Parameters are indexed by ***axes***.

Boolean Axis Status Parameters

Motion Status Parameters:

IN.MOT@	Servo(s) In Motion
IN.ACL@	Servo(s) Accelerating
AT.SPD@	Servo(s) at Constant Speed
IN.DCL@	Servo(s) Decelerating
AT.REST@	Servo(s) at Rest
IN.POS@	Servo(s) In Position
DSP.DONE@	No DSP motion changes in process or pending
OTL.FWD@ OTL.REV@	Forward / Reverse Overtravel Limit Status

DSP Inputs:

ZREF@	Zero Reference On
ASEN@	Sensor A On
BSEN@	Sensor B On
HTL.FWD@	Forward Limit Status
HTL.REV@	Reverse Limit Status

DSP Outputs:

ELS1@	Limit Switch 1
ELS2@	Limit Switch 2
ELS3@	Limit Switch 3
DELAY@	Internal Delay On

All **Boolean** Axis Status Parameters are indexed by ***axes***.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

INERTIA@	Servo Axis Inertia
INSPD.MUL@	Gear ratio input speed multiplier
IO.MODE@	Machine I/O configuration
KAF@	Acceleration Feedforward Factor
KP@	Position loop gain factor
KPI@	Position loop integral gain factor
KVF@	Velocity feedforward factor
KVH@	Velocity loop feedback gain factor
KVHA@	Analog velocity loop feedback gain factor
KVI@	Velocity loop integral gain factor
LOOP.RATE@	Servo loop update rate
MD.MODE@	MotionDATA mode
MODE@	Axis control mode
MTR.ACL.LIM@	Motor acceleration limit in rev/sec^2
MTR.SPD.LIM@	Axis speed limit in RPM
OTL.FWD@	Forward overtravel limit status
OTL.REV@	Reverse overtravel limit status
OUTSPD.MUL@	Gear ratio output speed multiplier
PB1@	Pushbutton PB1 pressed
PCR.SPD.LIM@	Pacer encoder speed limit in RPM
PCT.REV@	Pacer position counts/revolution
PERR.INPOS@	In-position error
PERR.MAX@	Maximum position error
PERR.NORM@	Normalization error
PG.LEN@	Page length used for the **LIST** command
POS.ABS@	Absolute encoder poll
POS.ACT@	Actual position
POS.ASEN@	Position at A-Sensor
POS.BSEN@	Position at B-Sensor
POS.CMD@	Position command
POS.DELAY@	Position at Delayed Sensor Input
POS.DIV@	Axis position units conversion divider
POS.ERR@	Current position error
POS.MOD@	Position modulus
POS.MUL@	Axis position units conversion multiplier
POS.ZREF@	Position at Encoder Reference
PPS.DIV@	Pacer position units divider
PPS.MUL@	Pacer position units multiplier
PW@	Program write enable
SCURVE@	S-Curve velocity profile
SENS.MODE@	Sensor Mode
SPD.MAX@	Maximum speed
STL.FWD@	Forward software travel limit
STL.REV@	Reverse software travel limit
TIME.MUL@	Time conversion factor
TRQ.GAIN@	Torque Gain of Motor and Drive
UNIT.ID@	Unit identification code
USER1.LED@	User 1 LED ON
USER2.LED@	User 2 LED ON
USR.ACL.LIM@	Acceleration limit in user units
USR.SPD.LIM@	Speed limit in user units
VEL.ACT@	Actual Velocity
VEL.CMD@	Velocity command
VEL.GAIN@	Velocity mode servo gain

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Syntax Errors

Code	Error Message
#1000	Syntax error
#1001	Invalid line number
#1002	Missing subscript
#1003	Read-only variable
#1004	**MOVE** syntax error
#1005	Error in data
#1006	Expected command
#1007	Garbage after statement
#1008	Too many parameters
#1009	Not enough parameters
#1010	Expected expression
#1011	Expected user variable name
#1012	Expected variable name
#1014	Expected a number
#1015	Expected line number or label
#1017	Expected comma
#1018	Expected comma or semicolon
#1019	Expected semicolon
#1020	Expected left paren
#1021	Expected right paren
#1022	Expected equal sign
#1023	Expected **GOTO**
#1024	Expected **THEN**
#1025	Expected **TO**
#1026	Expected ON, OFF or STOP
#1027	Expected a numeric variable
#1028	Expected an EVENT input signal
#1029	**GEAR** syntax error
#1031	Expected **IN** or **AT** to specify gear ratio
#1032	Expected a **string** variable

Syntax errors are caused by an incorrect, unexpected, or missing keyword, parameter, or delimiter. They are not caused by a value out of range or an incorrect variable type in an expression (see **Expression Errors**).

They can be fixed by modifying the line where the error occurred.

Note that non-trapped syntax errors will invoke the **line editor** with the cursor positioned at the point where the error was found.

Program Structure Errors

Code	Error Message
#1101	Missing equal sign or undefined line number/label
#1102	FOR without **NEXT**
#1103	NEXT without **FOR**
#1104	GOSUB without **RETURN**
#1105	RETURN without **GOSUB**
#1106	WHILE without **WEND**
#1107	WEND without **WHILE**
#1108	ELSE/ELSEIF without **IF-THEN** block
#1109	Missing **ENDIF**
#1110	**RESUME** without error
#1111	Out of data
#1112	Excess data
#1113	Function not found
#1114	Duplicate definition
#1115	**Block** structure must be the first statement
#1116	Missing **FIELD** statement
#1117	Statement cannot be nested
#1118	Statement not allowed inside a **FOR** loop

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

I/O Errors

Code	Error Message
#1301	File not found
#1302	Cannot create file
#1303	Input past end
#1304	Bad filename
#1305	Direct statement in file
#1306	Invalid baud rate
#1307	Cannot **open** device/file
#1308	Device/File already **open**
#1309	Device not **open**
#1310	Device write failed
#1311	Cannot **close** device
#1312	Device read failed
#1313	Illegal Discrete I/O Statement
#1314	Device not **opened** for writing
#1315	Device not **opened** for reading
#1316	**EIO.MODE@** not set for **PAMUX**

Code	Error Message
#1317	Timeout during **AIO@** access
#1318	Illegal analog I/O statement
#1319	Invalid **IO.MODE@** configuration
#1320	**IO.MODE@** for **PAMUX** is write only
#1321	Too many timed **DIO@** points active
#1322	No **Memory Disk** installed
#1324	**Memory Disk** is write protected
#1325	Invalid path
#1326	Disk full
#1327	**Memory Disk** is not formatted properly
#1328	File transfer aborted by **MotionPRO**
#1329	File save failed
#1330	**Memory disk** may contain data or **MotionCARD** in wrong slot
#1331	Unable to **FORMAT Memory Disk**
#1332	**Memory Disk** option is not supported
#1333	**MotionPRO**/MotionBASIC comm failure
#1334	Bad file mode

Code	Error Message
#1335	Device seek failed
#1336	ITM-270 Communications Failure
#1337	Path/File Access Error
#1338	Cannot **open** protected file
#1339	**Disk** is full

I/O errors are caused by an error during an attempted I/O operation. They are not caused by a syntax error in an I/O statement (see **Syntax Errors**).

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Error #1602: Accel or Decel out of range

Description: An attempt was made to perform a motion in which the specified acceleration or deceleration cannot be achieved.

Possible causes are:

- An acceleration or deceleration specified in a motion statement exceeds the values for **ACL.MAX@**, or **DCL.MAX@**, respectively.
- An acceleration or deceleration specified in a motion statement is not appropriate with the other specified parameters of that motion statement.

Error #1610: Illegal While DSP Busy

Description: An attempt was made to set the actual position, **POS.ACT@** while the DSP was busy. The DSP is busy when an accel or decel is being commanded, a **GEAR** is in effect, or there are motions waiting to execute in the **motion queue**.

System Errors

Code	Error Message	Reference
#1801	Out of memory	
#1802	Illegal direct	
#1803	Out of string space	**CLEAR**
#1804	Cannot continue	
#1805	Ctrl-C break detected	
#1806	Input buffer overflow	**EDIT**
#1807	Too many levels of nesting	
#1808	Illegal Indirect	
#1810	Program buffer write protected	**PW@**
#1811	Token buffer overflow	
#1812	Illegal while **FAULT@** is non-zero	
#1813	Illegal while **ESTOP.OK@** is **FALSE**	
#1814	Illegal while **AFAULT@** is non-zero	
#1815	MotionDATA not available	
#1816	Program buffer read protected	

System errors are caused by a system limitation (i.e. memory), incorrect mode, etc.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Error #1805: Ctrl-C Detected

Description: This error occurs when a Ctrl-C keystroke is read from the MotionPRO Development port.

MotionBASIC reacts differently to a Ctrl-C, depending on whether a program is running or not. If a CTRL-C is encountered when the controller is in **Direct Mode**, the following 4 steps are executed:

1. Clear the keyboard buffer
2. Bring motors to a controlled stop but leave them enabled
3. Clear all timed **DIO@** Points.
4. Reset the screen **COLOR** to white on black

If a CTRL-C is encountered when the Model 40 is running a program, the following steps will be executed:

1. Clear the keyboard buffer
2. Do normal Non-Fault Error processing which consists of:

   A) When an **Error Handler** routine **IS NOT** defined:

   - Set the **ERR** value to the **error code** number
   - Bring motors to controlled stop but leave them enabled.
   - Clear all timed **DIO@** Points
   - Turn on the **Program Error** LED
   - Reset the screen **COLOR** to White on Black
   - Print the error message
   - Return to **Direct Mode**

   B) When an **Error Handler** routine **IS** defined:

   - Set the **ERR** value to the error number
   - inhibit **EVENT** processing so the error handler runs "uninterrupted"
   - Start execution of the **ON ERROR** GOTO error handler.
   - (Note: When the **RESUME** is executed in the error handler routine, the EVENT processing will be re-enabled)

Error #1808: Illegal Indirect

Description: An attempt was made to execute a **direct**-only **command**, **indirectly**.

Error #1811: Token Buffer Overflow

Description: An attempt was made to download or enter a program line that when tokenized, overflows the 128 byte Token Buffer.

A line may be less than 128 characters, but is converted to tokens in MotionBASIC Storage. To remedy this error, shorten the program line.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Status, Faults & Error Codes

MotionBASIC provides very complete status and fault information and fail safe operation for a wide variety of possible problems, which include:

Controller Fault Errors	Problems with overall system operation
Error Codes	Description of all MotionBASIC error codes
Axis Modes	Modes of operation of individual axes
Axis Faults	Faults of individual axes in the system
Servodrive Alarms	Servodrive alarm codes for individual axes
System Status LEDs	Description of System Status LEDs & Diagnostics
Axis Module Failure	

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Power Fault - System Status LED

When the *Power Fault* LED is lit it indicates one of the following faults:

Battery Failure if the **Memory Fault** LED is flashing and the **System Watchdog OK** LED is flashing

Temporary Power Loss if the **Memory Fault** LED is NOT lit and the **System Watchdog OK** LED is NOT flashing

Servo Fault - System Status LED

When the *Servo Fault* LED is lit it indicates one of the following faults:

Axis Module Failure

Axis Fault

You can determine which of these faults occurred by checking the value of the **FAULT@** variable.

User 1 & User 2 - System Status LEDs

These two green LEDs are provided for user indications. They are turned off and on under program control by writing values to **USER1.LED@** and **USER2.LED@**.

User 1 LED also provides an important indication. It will flash during a **powerup** program load from a **Memory Disk**.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionDATA Communications Status LEDs

MotionDATA is a communications link for DSPs to share real-time motion information directly. It is a "non-bussed" RS-485 communications system which is connected in a "daisy chain" with modular 8-pin connectors and "line driver repeaters" on each unit.

The *MotionDATA Active*, *Sending MotionDATA* and *Receiving MotionDATA* System Status LEDs are provided for to assist the customer in troubleshooting any problems with MotionDATA communications.

MotionDATA Active	When lit, this LED indicates that MotionDATA is "active". i.e. The controller is receiving input on its MotionDATA clock line input.
Sending MotionDATA	When lit, indicates that the controller is transmitting MotionDATA communications.
Receiving MotionDATA	When lit, indicates that the controller is receiving MotionDATA communications.

Watchdog Timer - System Watchdog OK Status LED

Generation III controllers contain a "Watchdog Timer" which is located on the motherboard. This hardware circuitry consists of a timer which is directly connected to the **No Fault Relay** and will de-energize it unless it is reset by the multi-tasking operating system every 20 milliseconds.

For normal system operation, the *System Watchdog OK* status LED will be flashing at a rate of about once per second.

If controller software fails to respond within the 20 milliseconds, a **Watchdog Timeout** fault occurs, the *System Watchdog OK* LED **will not** be flashing as usual, and all the local **DIO@** outputs will be turned OFF.

If the normally-open contact of the **No Fault Relay** is interlocked with the servodrive Main Power as recommended in the "Installation & Operation Manual", then electromotive power will be removed from the servomotors if there is a firmware failure in the Generation III - Controller CPU.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

RAM Checksum Error

Each time the unit is "powered up" or "reset", the battery-backed **non-volatile memory** is tested by computing a checksum of the memory contents and comparing it with the former checksum.

If they are not equal, the red **Memory Fault** LED on the front of the controller will be flashing, a bit in the **FAULT@** variable will be set and automatic **powerup** program execution will be inhibited.

The red reset button on the front of the unit will NOT clear this fault, since the program or data in battery-backed RAM is corrupted and is only corrected by re-loading the program or editing the corrupted version in the RAM. Therefore, to clear a "RAM Checksum Error", change the contents of the program buffer using **LOAD**, **EDIT** or **NEW**. All MotionBASIC variable values in battery-backed RAM will be lost.

To clear the fault condition and reset the **No Fault Relay** you must explicitly clear the **FAULT@** variable.

Power Fault - Battery Failure

Each time the power is applied to the unit, there is a test of the lithium battery used to maintain information in the battery-backed RAM. If its voltage tests too low,

- the **Power Fault** LED on the front of the unit will be lit;
- the **Memory Fault** LED on the front of the unit will be flashing;
- the **System Watchdog OK** LED will be blinking;
- the **FAULT@** variable will be non-zero and the **No Fault Relay** off, and
- automatic **powerup** program execution will be inhibited.

As long as there is no **RAM Checksum Error**, pressing the red reset pushbutton on the front of the unit will clear this fault. This allows operation after powerup between time when the battery voltage tests low and the battery-backed RAM actually loses information during power-down.

If data is lost in the battery-backed RAM during power-down, the resulting **RAM Checksum Error** cannot be cleared by pressing the red reset pushbutton. Replace the battery promptly after the **first** battery failure error to prevent this loss of data during power-down.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Error #1911: Axis Fault

An axis (servo) fault is one of several fault conditions detected by a DSP Servo Axis Module, and reported by the **AFAULT@** variable. When an axis fault occurs, MotionBASIC issues a **controller fault error**, which disables the problem axis by setting its **MODE@** to zero. The **AFAULT@** and/or **ALARM@** variables are latched to indicate the cause of the fault. The **Servo Fault** LED on the front of the unit will be lit and the **No Fault Relay** will be "dropped out". The **AFAULT@**, **ALARM@**, **AXIS.FAULT@**, **AXIS.FLT1@** and **FAULT@** variables are set appropriately.

The statement AFAULT@ = 0 will attempt to clear the fault, but if the fault condition persists, AFAULT@ will remain non-zero. To enable the **No Fault Relay** after an axis fault condition is cleared, **FAULT@** must be explicitly cleared to 0 and **ESTOP.OK@** must be **TRUE**.

Please note that for errors which occur asynchronous to program operation, such as Axis Faults and **Overtravel Limits** or **E-Stop**, only the first occurrence will report a **user trappable error**. Additional reporting of these errors must be re-enabled with a **FAULT@**=0 statement.

Servodrive Alarm - Servo Fault

A servodrive alarm condition is detected when an axis is in **Mode** 2-5 and current is interrupted in the NO-ALARM' optically coupled input circuit at the DSP Axis Interface Connector (pin 23).

Unless NO-ALARM' is asserted, the **ALARM@** variable will have a value of 10 or more. The lower digit of the ALARM@ variable indicates the state of the three alarm code inputs AL03'-AL01' (pins 24-26).

ALARM@	**AL03'**	**AL02'**	**AL01'**	
0 or **10**	-	-	-	**value with no Axis Connector attached**
1 or 11	-	-	I	
2 or 12	-	I	-	The I's indicate current present in
3 or 13	-	I	I	the input circuit, which will be true
4 or 14	I	-	-	when the voltage at these connectors
5 or 15	I	-	I	is 4-25 volts lower than the V+ voltage
6 or 16	I	I	-	supplied at pin 22 of the Axis
7 or 17	I	I	I	Interface Connector.

Codes 0-7 are informational only and do not indicate an alarm condition.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Missing MotionDATA Communications - Servo Fault

This fault occurs when an axis performs a motion synchronized to **MotionDATA** and a packet of MotionDATA communications is not received at the **DSP Axis Module MotionDATA Interface.**

It can be generated by a broken wire or a missing MotionDATA communications cable.

See also: **MotionDATA not available pacer follower Electronic Gearing**

DSP Software Faults - Servo Fault

These faults occur when there is a software fault detected in the DSP software. If a DSP Software Fault occurs, please note its number and call **ORMEC Service** to report it.

Error #1912: Internal Error

An internal error is generated if at any time the software has found an internal malfunction of the Gen III Unit. The System Status LED, **System Watchdog OK** will not flash and the system will be "locked up".

If your controller develops an internal error, it is imperative that you call **ORMEC Service** to report this occurrence.

It is also advised that a "snapshot" of system memory be extracted from the unit while it is in this failure state. The "snapshot" of the failed state of the controller **AT THIS TIME** is important for problem analysis, especially if this occurrence is not easily repeatable.

This "snapshot" is called a **memory dump** during which the contents of the controller's memory is transmitted out of the RS-232 Serial Port to a file on the PC running **MotionPRO**.

See Also: **Memory Dump**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Error #1915: Incompatible MotionCARD Software

A **MotionCARD Fault** that indicates that the **MotionCARD** installed in the controller will not operate with the version of MotionBASIC installed in the controller.

One or the other **must be changed** to a compatible version.

A power cycle will be required to clear the error.

Error #1916: Not Enough Memory for MotionCARD

A **MotionCARD Fault** that indicates that the **MotionCARD** installed in the controller cannot operate with the current utilization of **non-volatile memory**.

There must be a minimum amount of space available in the User Comment space bank. The amount required will vary for different MotionCARD types. This error indicates that the user needs to change some program **comments** to be **disk comments** to free up more of this memory.

To clear this fault, you must:

1) Enable writing to the program buffer (**PW@**)
2) Execute the **NEW** command
3) Reset the controller fault variable. **FAULT@**=OFF

MotionCARD Fault

A **MotionCARD** fault is generated if at any time the software discovers a problem with the MotionCARD. Which particular problem exists with the MotionCARD would be detailed in the reported **Controller Fault Error**.

To clear these faults, you must:

1) Enable writing to the program buffer (**PW@**)
2) Execute the **NEW** command
3) Reset the controller fault variable. **FAULT@**=OFF

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Error #1999: Temporary Power Loss

Generation III controllers have circuitry which senses loss of incoming AC power before the +5 VDC computer power goes below specification. When incoming AC power loss is detected:

- the **Power Fault** LED is turned on;
- the **System Watchdog OK** light is not blinking; and
- all processing by the main CPU is suspended.

If the +5 VDC computer power supply goes below specification or power is cycled off and back on, then the CPU will be reset normally when +5 VDC power is restored and **ERR** will retain the value of the last error that occured before the power fault.

If AC power is restored before the +5 VDC computer power supply goes out of specification, the unit will be inoperative with the **Power Fault** light on, the **System Watchdog OK** light not blinking and **ERR**=1999.

This *Temporary Power Loss* fault may be cleared by pressing the red reset pushbutton on the front of the unit. Operation should proceed normally, with the exception that the **ERR** variable will be initially set to 1999.

No Fault Relay - No Fault LED

Generation III controllers have a "No Fault Relay" which is located on the I/O board on the front of the unit. This electro-mechanical relay has a 24 VDC coil which is powered by the internal 24 VDC power supply, and an isolated "normally-open" output contact.

As long as there is no fault condition in the unit, as defined by the bits in the **FAULT@** variable and **ESTOP.OK@** is **TRUE**, the No Fault Relay coil will be energized, holding the "normally-open" output contact closed. The green *NO FAULT* LED will be lit when and only when the relay coil is energized. The power to the No Fault Relay coil is also **hardware interlocked** with both the **E-STOP** input as well as the **Watchdog Timer** circuitry.

The fully isolated contact output is available to the user, and it is recommended that it be interlocked with the servodrive Main Power, which provides electromotive power to the servomotors. **To re-energize this relay after a fault,** ESTOP.OK@ **must be** TRUE **and the** FAULT@ **variable must be explicitly cleared.** These two conditions may be met in either order.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Full-Duplex Serial Communications

When using MotionPRO in "talk mode", pressing a key on the PC keyboard causes the PC to transmit a character to the controller, which in turn "echoes" that character to the screen of the PC. This communications can be observed as flashes on the **Receiving Serial** and **Sending Serial System Status LEDs**.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

ASCII TABLE

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80h	Ç	144	90h	É	160	A0h	á	176	B0h	▒
129	81h	ü	145	91h	æ	161	A1h	í	177	B1h	▓
130	82h	é	146	92h	Æ	162	A2h	ó	178	B2h	│
131	83h	â	147	93h	ô	163	A3h	ú	179	B3h	┤
132	84h	ä	148	94h	ö	164	A4h	ñ	180	B4h	╡
133	85h	à	149	95h	ò	165	A5h	Ñ	181	B5h	╢
134	86h	å	150	96h	û	166	A6h	ª	182	B6h	╖
135	87h	ç	151	97h	ù	167	A7h	º	183	B7h	╕
136	88h	ú	152	98h		168	A8h	¿	184	B8h	╣
137	89h	ë	153	99h	Ö	169	A9h	⌐	185	B9h	║
138	8Ah	è	154	9Ah	Ü	170	AAh	¬	186	BAh	╗
139	8Bh	ï	155	9Bh	¢	171	ABh	½	187	BBh	╝
140	8Ch	î	156	9Ch	£	172	ACh	¼	188	BCh	╜
141	8Dh	ì	157	9Dh	¥	173	ADh	¡	189	BDh	╛
142	8Eh	Ä	158	9Eh	₧	174	AEh	«	190	BEh	┐
143	8Fh	Å	159	9Fh	ƒ	175	AFh	»	191	BFh	└

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
192	C0h	┴	208	D0h	╨	224	E0h	α	240	F0h	≡
193	C1h	┴	209	D1h	╤	225	E1h	ß	241	F1h	±
194	C2h	┬	210	D2h	╥	226	E2h	Γ	242	F2h	≥
195	C3h	├	211	D3h	╙	227	E3h	π	243	F3h	≤
196	C4h	─	212	D4h	╘	228	E4h	Σ	244	F4h	⌠
197	C5h	┼	213	D5h	╒	229	E5h	σ	245	F5h	⌡
198	C6h	╞	214	D6h	╓	230	E6h	µ	246	F6h	÷
199	C7h	╟	215	D7h	╫	231	E7h	τ	247	F7h	≈
200	C8h	╚	216	D8h	╪	232	E8h	Φ	248	F8h	°
201	C9h	╔	217	D9h	┘	233	E9h	Θ	249	F9h	∙
202	CAh	╩	218	DAh	┌	234	EAh	Ω	250	FAh	·
203	CBh	╦	219	DBh	▀	235	EBh	δ	251	FBh	√
204	CCh	╠	220	DCh	▄	236	ECh	∞	252	FCh	ⁿ
205	CDh	═	221	DDh	▌	237	EDh	φ	253	FDh	²
206	CEh	╬	222	DEh	▐	238	EEh	ε	254	FEh	■
207	CFh	╧	223	DFh	▀	239	EFh	∩	255	FFh	

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionBASIC Key Table Keys with several MotionBASIC uses.

Key	INKEY$	Fncode	KEY (n)	EXITKEY	INPUT @ Effect
ESC	CHR$(27)			32	Unchanged
ENTER	CHR$(13)			0	Changed
F1	CHR$(0)+CHR$(59)	59	1	1	"
F2	CHR$(0)+CHR$(60)	60	2	2	"
F3	CHR$(0)+CHR$(61)	61	3	3	"
F4	CHR$(0)+CHR$(62)	62	4	4	"
F5	CHR$(0)+CHR$(63)	63	5	5	"
F6	CHR$(0)+CHR$(64)	64	6	6	"
F7	CHR$(0)+CHR$(65)	65	7	7	"
F8	CHR$(0)+CHR$(66)	66	8	8	"
F9	CHR$(0)+CHR$(67)	67	9	9	"
F10	CHR$(0)+CHR$(68)	68	10	10	"
↑	CHR$(0)+CHR$(72)		11	11	"
←	CHR$(0)+CHR$(75)		12	12	"
→	CHR$(0)+CHR$(77)		13	13	"
↓	CHR$(0)+CHR$(80)		14	14	"

Keyboard Scan Codes See: **INKEY$**

Key	Scancode	Key	Scancode	Key	Scancode
F1	59	↓	80	Ctrl F3	96
F2	60	PgDn	81	Ctrl F4	97
F3	61	Ins	82	Ctrl F5	98
F4	62	Del	83	Ctrl F6	99
F5	63	Shift F1	84	Ctrl F7	100
F6	64	Shift F2	85	Ctrl F8	101
F7	65	Shift F3	86	Ctrl F9	102
F8	66	Shift F4	87	Ctrl F10	103
F9	67	Shift F5	88	Ctrl Home	119
F10	68	Shift F6	89	Ctrl PgUp	132
Home	71	Shift F7	90	Ctrl ←	115
↑	72	Shift F8	91	Ctrl →	116
PgUp	73	Shift F9	92	Ctrl End	117
←	75	Shift F10	93	Ctrl PgDn	118
→	77	Ctrl F1	94	Ctrl 2	3
End	79	Ctrl F2	95	Alt 1	120

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

DELAY.DIST@
DELAY@
DELETE
DIM
DIO@
DIST.BIAS@
DIST.GEAR@
DIST.GRAT@
DIST.MOVE@
DSP.CTR@
DSP.DONE@
DRV.CMD@
DRV.MAX@
DUMP
EDIT
EIO.MODE@
ELS1@
ELS1.HIGH@
ELS1.LOW@
ELS2@
ELS2.HIGH@
ELS2.LOW@
ELS3@
ELS3.HIGH@
ELS3.LOW@
ELSE
ELSEIF
END
ENDIF
EOF
ERASE
ERL
ERR
ERR$
ERROR
ESTOP.OK@
EVENT
EXITKEY
EXP
FALSE
FAULT@
FIELD
FILES
FIX
FN
FOR
FORMAT
FRE
GEAR
GEAR.RATIO@
GET
GOSUB
GOTO
HALT
HEX$
HTL.FWD@
HTL.REV@
IF
IN
IN.ACL@
IN.DCL@
IN.MOT@
IN.POS@
INERTIA@
INKEY$
INP
INPUT
INPUT #
INPUT @
INPUT$
INPW
INSPD.MUL@
INSTR
INT
IOCTL
IO.MODE@
KAF@
KEY
KEY (n)
KILL
KP@
KPI@
KVF@
KVH@
KVHA@
KVI@
LCASE$
LEFT$
LEN
LET
LIST
LOAD
LOC
LOCATE
LOF
LOG
LOOP.RATE@
LSET
MBVER$
MD.MODE@
MERGE
MID$
MKDIR
MOD
MODE@
MOVE
MP.CONFIG
MTR.ACL.LIM@
MTR.SPD.LIM@
NAME
NEW
NEW RESTORE
NEXT
NOT
OCT$
OFF
ON
ON ERROR
ON EVENT
ON KEY
ON...GOSUB
ON...GOTO
OPEN
OR
OTL.FWD@
OTL.REV@
OUT
OUTSPD.MUL@
OUTW
PB1@
PB1LOAD.BAS
PCR.SPD.LIM@
PCT.REV@
PEEK
PEEKW
PERR.INPOS@
PERR.MAX@
PERR.NORM@
PG.LEN@
POKE
POKEW
POS
POS.ABS@
POS.ACT@
POS.ASEN@
POS.BSEN@
POS.CMD@
POS.DELAY@
POS.DIV@
POS.ERR@
POS.MOD@
POS.MUL@
POS.ZREF@
PPS.DIV@
PPS.MUL@
PRINT
PRINT #
PUT
PW@
RANDOMIZE
READ
REM
RENUM
REPEAT
RESTORE
RESUME
RETURN
RIGHT$
RMDIR
RND
RSET
RUN
SAVE
SCURVE@
SEG
SENS.MODE@
SGN
SIN
SPACE$
SPC
SPD.MAX@
SQR
STACK
STEP
STL.FWD@
STL.REV@
STOP
STR$
STRING$
SYSTEM
TAB
TALK
TAN
THEN
TIME$
TIME.MUL@
TIMER
TO
TRACE
TRACE VAL
TRQ.GAIN@
TRUE
UCASE$
UNIT.ID@
UNTIL
USER1.LED@
USER2.LED@
USING
USR.ACL.LIM@
USR.SPD.LIM@
VAL
VEL.ACT@
VEL.CMD@
VEL.GAIN@
VEL.LAST@
VLTC@

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Purpose: Adds explanatory text to a program listing. See also ''.

Syntax: '[*remark*]

remark Commentary text, ignored by MotionBASIC.

MotionBASIC does not execute comments, but a program may branch to a statement beginning with '. Program operation continues with the first executable statement thereafter.

An ' can also be used to add a comment at the end of any program line, which **REM** cannot. MotionBASIC will ignore everything from the ' character to the end of the line.

Note that the first 20K bytes of comments are stored in a separate section of the **memory map**, so that they reduce the 60K bytes used for program and variable storage by only three characters per comment. Note also that characters which appear consecutively more than three times

are stored in compressed format to minimize storage requirements. e.g. '---------- will take only six bytes of comment space.

Example:

```
10 A=100                    'Set the initial acceleration
20 V=48                     'Set the initial velocity
30 PRINT "A=";A;"  V=";V    'Display the acceleration & velocity
40 END
RUN
A=100   V=48
```

See Also: '', **Disk Comment**, **REM**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

'!DEFINE **Conditional Loading Symbol Definition** Directive

Purpose: Define a symbol used for conditional loading in a program file.

Syntax: '!DEFINE **symbol**

symbol The symbol name to define. Symbol names may contain the same characters as **variable names**, and may be 32 characters long.

Symbols defined in the loading program file are also defined for all included files. See: **'!INCLUDE**

See Also: **'!UNDEF** **'!IFDEF** **'!IFNDEF**

'!IFDEF / '!IFNDEF Conditional Loading Block Construct Directive

Purpose: Conditionally load a section of the **program file** if a symbol is previously defined (or not defined).

Syntax:

```
'!IFDEF symbol                    '!IFNDEF symbol
   [statements]                      [statements]
['!ELSE]                          ['!ELSE]
   [statements]                      [statements]
'!ENDIF                           '!ENDIF
```

symbol The symbol name to check for its definition state.
statements One or more code lines (may contain MotionPRO directives).

IFDEF is TRUE if the symbol is defined.
IFNDEF is TRUE if the symbol is not defined.

IF[N]DEF...ELSE...ENDIF structures can be nested any number of times.

See also: **'!DEFINE** **'!UNDEF**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

+ Addition, Concatenation, or Set Union Operator

Purpose: Perform addition, string concatenation, or set union.

Usage1: *exp1* + *exp2* Addition

Usage2: *str1* + *str2* String Concatenation

Usage3: *set1* + *set2* Set Union

exp1, *exp2* Numeric expressions which represent the two values to add.

str1, *str2* String expressions. The result of the concatenation is a string composed of *str1* followed by *str2*.

set1, *set2* Set expressions. The result of the union is the set of all elements which are in either *set1* and *set2*.

If the expression involves numeric operands of more than one precision, all operands are converted to level of the highest-precision operand in the expression before the expression is evaluated.

See also: - * / \ **MOD** ^ **Precedence** **Garbage Collection**

- Subtraction, Negation, or Set Difference Operator

Purpose: Perform subtraction, negation, or set difference.

Usage1: *exp1* - *exp2* Subtraction

Usage2: - *exp* Negation (unary)

Usage3: *set1* - *set2* Set Difference

exp1, *exp2* Numeric expressions which represent the two values to subtract.

exp A numeric expression which represents the number to negate (reverse the sign of).

set1, *set2* Set expressions. The result is the set of all elements in *set1* that are not elements in *set2*.

If the expression involves operands of more than one precision, all operands are converted to the level of the highest-precision operand in the expression before the expression is evaluated.

See also: + * / \ **MOD** ^ **Precedence** **Differences**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

<= Less than or Equal to, or Set Membership Operator

Purpose: Perform a "less than or equal to" relational comparison, or set membership.

Usage1: *exp1* <= *exp2* Less Than or Equal

Usage2: *set1* <= *set2* Set Membership

exp1, exp2 Numeric or string expressions. Both *exp1* and *exp2* must be either numeric or string. If the value of *exp1* is less than or equal to that of *exp2*, the result is -1 (TRUE); otherwise the result is 0 (FALSE).

set1, set2 Set expressions. If *set2* has all the elements in *set1*, the result is -1 (TRUE); otherwise the result is 0 (FALSE).

In **IF** statements, BASIC regards 0 as equivalent to FALSE and all non-zero values as equivalent to TRUE.

If the operands are string expressions, BASIC compares the **ASCII** values of the first character in each expression. If the first characters have the same ASCII value, BASIC then compares the second characters, and so on. If one string terminates before the other, and the characters in each string are identical to that point, the shorter string is considered "less" than the longer one.

If *exp1* and *exp2* are not both numeric or both strings, a Type Mismatch error will occur.

See also: **= <> < > >= WITHIN Precedence**

Example:
```
10 IF A <= B THEN PRINT "A is less than or equal to B"
```

<> Not Equal to Operator

Purpose: Perform a "not equal to" relational comparison.

Usage1: *exp1* <> *exp2* Not Equal To

Usage2: *set1* <> *set2* Set Membership

exp1, exp2 Numeric or string expressions. Both *exp1* and *exp2* must be either numeric or string. If the value of *exp1* is not equal to that of *exp2*, the result is -1 (TRUE); otherwise the result is 0 (FALSE).

set1, set2 Set expressions. If *set1* is not identical to *set2*, the result is -1 (TRUE); otherwise the result is 0 (FALSE).

In **IF** statements, BASIC regards 0 as equivalent to FALSE and all non-zero values as equivalent to TRUE.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

= Assignment or Equal — Operator

It is not good programming style to use **single** or **double** precision floating point **variables** with the = operator. A comparison may fail because of the inexactness of the numeric representation.

See also: **<>** **<** **>** **<=** **>=** **WITHIN** **Precedence** **Garbage Collection**

Example 1: Assignment

```
10 A = C + 1                'Assign the value of "C + 1" to variable A
```

Example 2: Equal to Operator

```
1020 IF A = B THEN PRINT "A is equal to B"
```

> Greater than, or Set Membership — Operator

Purpose: Perform a "greater than" relational comparison, or set membership.

Usage1: *exp1* > *exp2* — Greater Than

Usage2: *set1* > *set2* — Set Membership

exp1, *exp2* — Numeric or string expressions. Both *exp1* and *exp2* must be either numeric or string. If the value of *exp1* is greater than that of *exp2*, the result is -1 (TRUE); otherwise the result is 0 (FALSE).

set1, *set2* — Set expression. If *set1* has all the elements in *set2* and *set1* is not equal to *set2*, the result is -1 (TRUE); otherwise the result is 0 (FALSE).

In **IF** statements, BASIC regards 0 as equivalent to FALSE and all non-zero values as equivalent to TRUE.

If the operands are string expressions, BASIC compares the **ASCII** values of the first character in each expression. If the first characters have the same ASCII value, BASIC then compares the second characters, and so on. If one string terminates before the other, and the characters in each string are identical to that point, the longer string is considered "greater" than the shorter one.

If *exp1* and *exp2* are not both numeric or both strings, a Type Mismatch error will occur.

See also: **=** **<>** **<** **<=** **>=** **WITHIN** **Precedence**

Example:

```
10 IF A > B THEN PRINT "A is greater than B"
```


Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

\ Integer Division — Operator

Purpose: Perform integer division.

Usage: *exp1* \ *exp2*

exp1 An integer expression which represents the dividend.

exp2 An integer expression which represents the divisor. The divisor must not be zero.

The operands must be either **integer** or **long integer**. Any other operand types will cause a Type Mismatch error to occur. The result is the same type as the largest operand. If the two operands are different sizes, **integer type conversion** is performed.

See also: **+ - * / MOD ^ Precedence**

^ Exponentiation — Operator

Purpose: Perform exponentiation.

Usage: *exp1* ^ *exp2*

exp1 A numeric expression which represents the value to be raised to a power. This may be negative if and only if *exp2* is an integer.

exp2 A numeric expression which represents the power to raise *exp1* to. This may be fractional or negative. It must be an integer if *exp1* is negative.

Both operands are converted to **double precision** floating-point before the operation takes place, and the result is always double precision floating-point.

See also: **+ - * / \ MOD Precedence Differences**

ABS Absolute Value — Function

Purpose: Return the absolute value of an expression.

Usage: ***var*** = ABS(***numexpr***)

ACL.CMD@ Acceleration Command — ORMEC Variable

Purpose: Return the current real-time value for commanded acceleration (or deceleration) rate of a servo axis in user units.

Usage: ***var&*** = ACL.CMD@[(***axes***)]

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

AFAULT@ **Axis Fault Code** ORMEC Variable

AXIS.FAULT@ identifies the **set** of all servo axes in the system that currently have a fault condition (other than zero).

AXIS.FLT1@ indicates the first axis to detect a fault.

When an **axis fault** occurs, MotionBASIC will automatically set **MODE@** to 0 for the problem axis and issue a **controller fault** error. The **AFAULT@** variable will be latched to a non-zero value which indicates the cause of the fault. The statement AFAULT@ = 0 will attempt to clear the fault, but if the fault condition persists, AFAULT@ will remain non-zero.

To enable the **No Fault Relay** after an axis fault condition is cleared, **FAULT@** must be explicitly cleared to 0 and **ESTOP.OK@** must be **TRUE**.

AFTER **Define a Start Condition** Modifier

Purpose: Define a start condition for a motion specified by a **MOVE**, **GEAR** or **HALT** statement; define a start condition for a **WAIT** statement.

Usage 1: [REPEAT] **MOVE** *parameters* AFTER *condition* [UNTIL *condition*]

Usage 2: [REPEAT] **GEAR** *parameters* AFTER *condition* [UNTIL *condition*]

Usage 3: **HALT** *parameters* AFTER *condition* [UNTIL *condition*]

Usage 4: **WAIT** *time* [AFTER *expression*]

condition **ZREF@**, **ASEN@**, **BSEN@** or **DELAY@**

expression Any **Boolean** expression.

When MotionBASIC executes a **MOVE**, **GEAR** or **HALT** statement, it processes the parameters and places high level information in the **shared memory** of the **DSP Axis Module**(s). When *AFTER*, **UNTIL** and/or **REPEAT** modifiers are

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

AIN2@ **Axis Analog Input Two** ORMEC Variable

Purpose: Read a 12-bit **analog input** at the **Axis Interface Connector** of the **DSP Axis Module.**

Usage: ***var%*** = AIN2@[(***axes***)] Read Analog Input/Output

var% Returns an **integer** value in the range -32768 to +32767, which corresponds to an analog input voltage from -10 to +10 VDC at the **optional** AIN2 analog input. **This variable will return unpredictable values if the 12-bit analog input option is not present on the** DSP Axis Module!

Example:

```
10 X% = AIN2@({3})      'read a 12-bit analog input for axis 3
20 PRINT X%             'print the value read
30 END
RUN
-32768                  'printed if the value read was -10 VDC
```

See also: **AIN1@** **Loop Architecture**

AIO@ **Analog I/O Point** ORMEC Variable

Purpose: Allow MotionBASIC to read or set **Analog I/O** points using the **Pamux** Interface.

Syntax: AIO@(*point*) = *value* Write Analog Output

Usage: ***var%*** = AIO@(*point*) Read Analog Input/Output

point An **integer** expression in the range from 17 to 528 inclusive.

value An **integer** expression in the range from 0 to 4096 when writing to a single-ended analog output and in the range from -2048 to 2047 when writing to a bipolar output.

var% An **integer** value will be returned in the range from 0 to 4096 when reading a single-ended analog I/O point, and in the range from -2048 to 2047 when reading a bipolar analog I/O point. When an analog output is read, the value

returned will be the value last written to it. The default value for analog outputs is 0.

The AIO@ array variable is only accessible when **EIO.MODE@** is configured for **Pamux** and the type and direction of each AIO point has been configured using **IO.MODE@**.

Example:

```
10 AIO@(33) = 512
20 X = AIO@(34)
```

The program lines above set the level of Analog Output point 33 to 512 and assign the value of Analog I/O point 34 to a variable named 'X'.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Note that when the **AFAULT@** variable is cleared after a servodrive ALARM condition, the controller asserts the Alarm Reset Output (ALM-RESET') at the **DSP Axis Module**'s Axis Interface Connector for up to 50 msec in an attempt to clear the servodrive ALARM condition.

S-Series servodrives do not support an external reset signal and must be reset by pressing the pushbutton on the front of the drive near the "Drive Fault Indicator". If S-Series servodrives must be reset automatically, then you must cycle the servodrive control power using a discrete digital output. See **DIO@**.

F-Series servodrives require a power cycle to clear:
High Bus Voltage Fault (ALARM@ = 12 or 16)
Drive Current Output Fault (ALARM@ = 13 or 17)
All other faults can be cleared by either a power cycle, or by clearing the **AFAULT@** variable.

See also: **FAULT@** **AFAULT@** **AXIS.FAULT@** **AXIS.FLT1@**

AND **Logical AND** Operator

Purpose: Perform a bitwise logical AND on two binary integers or long integers.

Usage: *exp1* AND *exp2*

exp1, exp2 Any expressions, including **boolean** expressions, that produce an integer or long integer result.

The operands must be either **integer** or **long integer**. Any other operand types will cause a Type Mismatch error to occur. The result is the same type as the largest operand. If the two operands are different sizes, **integer type conversion** is performed.

MotionBASIC performs a **bitwise** AND, according to the AND **truth table**. See the chapter on **bitwise operations** for an example.

See also: **NOT** **OR** **XOR** **Precedence**

ASC **ASCII Value of First Character** Function

Purpose: Return the **ASCII** value of the first character of a string expression.

Usage: ***var%*** = ASC(*stringexpr*)

stringexpr A string expression.

Note: If the argument is a null string, ASC generates an Empty String error.

See also: **CHR$** **VAL**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

ATN **Arctangent** Function

Purpose: Return, in radians, the angle whose tangent is *numexpr*. The return value is in the range -PI/2 to +PI/2.

Usage: ***var#*** = ATN(***numexpr***)

Note: PI radians = 180 degrees, and so to convert from radians to degrees, multiply the returned value by 180/PI or 57.296.

See also: **COS** **SIN** **TAN**

AUTO **Automatic Line Numbering** Command

Purpose: Activate automatic line numbering for keying in a program with the MotionBASIC **line editor**, starting at a specified line number and incrementing by a specified amount. See also **'!AUTO**.

Syntax: AUTO [*startline* [,*increment*]]

startline The starting line number or a label defining the starting line number. See **PW@** for the default values.

increment The interval between line numbers. Defaults to 10.

If AUTO is about to write a line number that already exists, an asterisk (*) is displayed next to the number. To avoid overwriting the existing line, and return to direct mode, press the **Enter** key immediately after the asterisk.

Since AUTO calls the **line editor** for input, the line **edit keys** may be used during line entry. If the "up-arrow" key is pressed, you may

also **EDIT** previous lines. When the cursor is on the last program line entered, pressing the **Enter** key will resume AUTO numbering at the next appropriate line number.

To terminate automatic line numbering, press Enter on an empty line or "break" the command by pressing Ctrl-C.

Automatic Load Program File

AUTOLOAD.BAS is included in the list of **keywords**, but it is not a keyword in the strictest sense. It is the **required** filename for the program file which is designated to automatically load into the controller from the **Memory Disk** on **powerup**.

This file must be stored in the root directory of the **Memory Disk** to provide this functionality.

See Also: **PB1LOAD.BAS**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

AXIS.LIST@ **Set of all servos in the system** ORMEC Variable

Purpose: Identify all of the servo axes in the system. Remove one or more axes from the system.

Syntax: AXIS.LIST@ = *set*

Usage: ***var~*** = AXIS.LIST@

set The **set** of all servos in the system. All servo axes specified in this expression must be included in the current AXIS.LIST@. In other words, axes may be removed from AXIS.LIST@ but may never be added. Default AXIS.LIST@ = all servo axes found at powerup.

var~ A **set** variable which is to receive the **set** of all servo axis found.

Example:

```
100 AXIS.LIST@ = AXIS.LIST@ - {2}      'Remove axis 2 from system
```

AXIS.OTL@ **Set of Servos in the System with an OTL** ORMEC Variable

Purpose: Identify the **set** of servo axes which currently have an overtravel limit condition. i.e. either OTL.FWD@ or OTL.REV@ **TRUE**

Usage: ***var~*** = AXIS.OTL@

var~ A **set** variable which is to receive the set of all servos in the system with an overtravel limit condition.

AXIS.SET@ **Default Set of Servos** ORMEC Variable

Purpose: Define a default **set** of servos to MotionBASIC. This default set of servos will be used if no ***axes*** parameter is specified in statements or variables where it is optional.

Syntax: AXIS.SET@ = *set* Define the Default Axis Set

Usage: ***var~*** = AXIS.SET@ Get the Default Axis Set

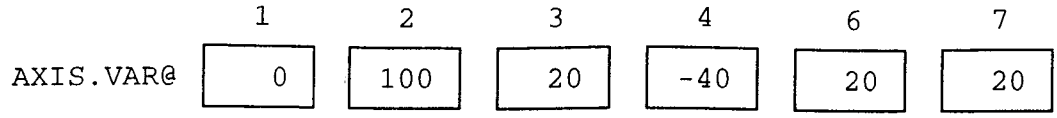
set The **set** of servos to be used by default. All servo axes specified in this expression must be included in **AXIS.LIST@**. Default AXIS.SET@ = AXIS.LIST@.

var~ A set variable which is to receive the current default axis set.

See the Example below.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

AXIS.VAR@ **Axis-Indexed Long Integer Array** ORMEC Variable



Now consider the operation of the following statements:

```
AXIS.SET@ = {2,3}
MOVE FOR AXIS.VAR@ IN AXIS.VAR@          'command two axes to "index"
'Axis 2 moves an incremental 100 position units during 100 time units
'Axis 3 moves an incremental  20 position units during  20 time units

MOVE AT -AXIS.VAR@      'Axis 2 ramps to the speed -100
                        'Axis 3 ramps to the speed -100
'The indexing feature does not apply because the - changed AXIS.VAR@ to
'an expression.  -AXIS.VAR@ was evaluated as -AXIS.VAR@({2,3}) or -100.
```

The specific motion parameters that can be indexed to AXIS.VAR@ elements are indicated (with a "♦") in the **MOVE**, **GEAR**, & **HALT** Syntax Charts.

BAUD@ **Serial Port Baud Rate** ORMEC Variable

Purpose: Return the baud rate of the RS-232 serial port, SRL1, which is dedicated to motion program development and troubleshooting.

Syntax: BAUD@ = 0 'Initiate autobaud sequence
Usage: ***var***& = BAUD@ 'return the current Serial Port baud rate

var An **integer** variable which will have one of the following values (2400, 4800, 9600, 19200, 38400) indicating the current baud rate of the Serial Port used for **MotionPRO** development.

The statement "BAUD@ = 0" initiates an autobaud sequence which requires that the next two characters be **ASCII** "carriage returns" (Code **CR**) which are sent a minimum of 10 msec apart. When the first one is received, the baud rate is selected from the possible values above. The second one is used to verify that the proper baud rate was selected. This sequence is automatically executed by MotionPRO when a baud rate is selected from the menu.

BIN$ **Binary Value, as String** Function

Purpose: Return, as a string, the binary representation of a number.

Usage: ***var$*** = BIN$(***lngexpr***)

If *lngexpr* is negative, BIN$ returns the twos complement value in binary.

The returned string contains no leading zeros.

See also: **HEX$** **STR$** **OCT$**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

BSEN@ **B-Sensor On** ORMEC Variable

Purpose: Allow MotionBASIC to determine if one of the high speed sensors on the **DSP Axis Module** (B-Sensor) is asserted; or has been asserted since last cleared. Clear and arm an edge sensitive latch.

Usage: ***var%*** = BSEN@[(***axes***)] 'w/ multiple axes reference, BSEN@s are ORed
Syntax: BSEN@[(***axes***)] = 0

var% A **Boolean** value which will be TRUE (-1) if B-Sensor either is asserted, or has been asserted since last cleared, depending on **SENS.MODE@**. Otherwise it will be FALSE (0).

This input signal is used to capture axis position **POS.BSEN@**. BSEN@ can initiate a motion **AFTER** it occurs, terminate a motion waiting **UNTIL** it occurs, and it can be used in any **boolean** expression.

Setting BSEN@ to 0 always clears it if it was configured by **SENS.MODE@** as **R**ising or **F**alling edge sensitive. If configured level sensitive, it may be immediately re-asserted depending on the state of the input.

BYE **Return to Operating System (BASIC)** Command

Purpose: **None. This keyword is currently reserved for future use.**

CHAIN **Load and Execute Program or Overlay** Statement

Purpose: The CHAIN Command is used to transfer control to the specified program from a **Memory Disk** or from a PC running **MotionPRO** without affecting the variable values.

Syntax: CHAIN [MERGE] *filespec* [,{*linum* | *label*}] [,DELETE *start*][[-]*end*]]]

filespec "[*drive*:][*path*\]*filename*[.*ext*]"

drive Development Computer drive letter or **1** for **Memory Disk**.
path A string expression that follows **DOS file-naming**
filename conventions.
ext Default extension is .BAS.

linum The numeric literal that specifies the line number at which program execution should resume after the file has been loaded. If no line number is specified, execution will begin at the first line in the program buffer.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

CHR$ **Convert ASCII Code to Character** Function

Purpose: Return the character whose **ASCII** value is specified.

Usage: ***var$*** = CHR$(*code*)

code An **integer expression** in the range 0 to 255 which corresponds to the decimal value in the **ASCII** chart.

See also: **ASC** **STR$**

CLEAR **Reset Variables; Set Maximum String Space** Statement

Purpose: Erase all user **variables** and set all predefined **axis variables** to their powerup defaults; or specify the maximum size of MotionBASIC's string space.

Usage: CLEAR [*strsize*]

strsize Maximum size, in bytes, of MotionBASIC's string space. Defaults to 1024.

CLEAR erases all variable data, including any information set via **DEF FN**, **DEF SEG**, **DEFINT**, **DEFLNG**, **DEFDBL**, **DEFSNG**, **DEFSTR** and **DEFSET**. Unlike **NEW**, however, it leaves the current program intact.

See also: **ERASE** **NEW** **Program Initialization** **Garbage Collection**

CLG **Common Logarithm** Function

Purpose: Return the common (base 10) logarithm of a number.

Usage: ***var#*** = CLG(***numexpr***)

numexpr A numeric expression greater than 0.

See also: **LOG** **EXP**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

CNT.REV@ **Axis Counts/Rev** ORMEC Variable

Purpose: Define the number of counts per revolution of the servo axis position transducer.

Syntax: CNT.REV@[(***axes***)] = *counts*

Usage: ***var%*** = CNT.REV@[(***axes***)]

counts A numeric expression in the range 1 to 32768, which represents the number of counts per revolution of the **axis encoder**. Default CNT.REV@ = 6000.

var% A numeric variable to receive the current number of counts per revolution of the servomotor as defined by CNT.REV@.

This parameter should be set prior to setting any other **user units**.

MAC-E Series Servomotors:	6,000 counts per rev
MAC-A Series Servomotors:	6,000 counts per rev
MAC-B Series Servomotors:	24,000 counts per rev
MAC-C Series Servomotors:	24,000 counts per rev
MAC-F Series Servomotors	
with F-Series "/A" drives:	16,384 counts per rev
with F-Series "/B" drives:	4,096 counts per rev

COLOR **Set Foreground and Background Colors** Statement

Purpose: Specify foreground and background colors for the video display.

Usage: COLOR [*foreground*] [,[*background*]

foreground An integer expression from 0 to 31 which selects the foreground color from the table below.

background An integer expression from 0 to 7 which selects the background color from the table below.

At least one of the optional parameters above must be included. Omitted parameters retain former values. The default is white foreground (7) and black background (0).

See also: **Differences**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

CSRLIN **Row Position of Cursor** Function

Purpose: Return the row number of the current cursor location on the current screen page.

Usage: ***var%*** = CSRLIN

To get the current column position, use **POS**. To set the cursor position, use **LOCATE**.

CW.FWD@ **Clockwise Forward** ORMEC Variable

Purpose: Cause the servomotor's direction of "forward" travel to be "clockwise", as viewed from the motor shaft end of the motor.

Syntax: CW.FWD@[(***axes***)] = *mode*

Usage: ***var%*** = CW.FWD@[(***axes***)]

mode A **Boolean** expression that, when non-zero, causes the motor to rotate clockwise when commanded to move in a positive direction.

var% A numeric variable to receive the current "forward" direction as defined by CW.FWD@.

Default forward rotation of the motor is counter-clockwise (CW.FWD@=0).

See also: **DRV.CMD@** **DRV.MAX@**

DATA **Store Constant(s) for Retrieval via READ** Statement

Purpose: Store one or more constants, of any type, for subsequent access via **READ** statements.

Syntax: DATA *constant1* [,*constant2*]...

DATA statements are nonexecutable statements that supply a stream of constants for use by **READ** statements. All the items supplied by all the DATA statements in a program make up one continuous string of information that is accessed in order by your program's READ statements. Use **RESTORE** to reset the pointer to the first item in a specified DATA statement.

You may put as many constants (expressions are not allowed) in a single DATA statement as will fit on one line, and your program may have any number of DATA statements, positioned anywhere in the program.

String constants in DATA statements do not need to be surrounded by quote marks, unless they contain quote marks themselves, colons, commas, single-quotes, or significant leading or trailing blanks.

See also: **Differences** from GWBASIC **Program Initialization**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

DCL.MAX@ **Maximum Deceleration** ORMEC Variable

Purpose: Define the maximum allowable commanded deceleration rate in **user units**. This is also the default deceleration rate used by **motion statements**.

Syntax: DCL.MAX@[(***axes***)] = *decel*

Usage: ***var&*** = DCL.MAX@[(***axes***)]

decel A numeric expression representing the maximum allowable commanded axis deceleration rate in user units from 1 to **USR.ACL.LIM@**. A value of 0 disables maximum deceleration checking, allowing instant decelerations to be commanded. Default = USR.ACL.LIM@, and **changing** USR.ACL.LIM@ **will always reset DCL.MAX@ equal to USR.ACL.LIM@.**

Any motion statement that does not have an deceleration rate specified in some way will use this rate. If a motion statement specifies an deceleration rate greater than this parameter, an error will be generated and the motion will not occur.

DEF FN **Define Function** Statement

Purpose: Define a user function.

Syntax: DEF FN*name*[(*parmlist*)] = *expression*

name A valid variable name (with type suffix). The suffix establishes the **data type** of the value returned. Defaults to integer type (%) when suffix is omitted.

parmlist One or more parameters for the function. Each must be a valid variable name. Parameters are passed by value.

expression An expression, of the same type as *name*, that defines the value returned by the function.

Function definitions may include any number of arguments as dummy parameters. The arguments serve as definition placeholders only; even if they happen to be the same as variable names currently in use within the program, they do not affect the values of those variables.

A function definition is limited to a single MotionBASIC statement and it must be executed before the function is called.

Functions may not call themselves.

To call a function, use the syntax

```
v = FNname [(parameter [,parameter]...)]
```

DEF FN may not be used in direct mode.

See also: **Program Initialization**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

DEFLNG **Define Variable(s) as Long Integer** Statement

Purpose: Declare one or more variables to be of type **Long Integer.**

Syntax: DEFLNG *letter*[-*letter*]

All variable names beginning with any of the specified *letters* (case-insensitive) will be Long Integer, unless a type identifier character (%, #, !, $, ~, or @) is used to override the DEFLNG declaration.

The DEFLNG declaration must be processed before any of the declared variables are used.

See also: **DEFINT** **DEFDBL** **DEFSET** **DEFSNG** **DEFSTR**

DEFSET **Define Variable(s) as Set** Statement

Purpose: Declare one or more variables to be of type **set.**

Syntax: DEFSET *letter*[-*letter*]

All variable names beginning with any of the specified *letters* (case-insensitive) will be Set, unless a type identifier character (%, &, #, !, $, ~, or @) is used to override the DEFSET declaration.

The DEFSET declaration must be processed before any of the declared variables are used.

See also: **DEFINT** **DEFLNG** **DEFDBL** **DEFSNG** **DEFSTR**

DEFSNG **Define Variable(s) as Single Precision** Statement

Purpose: Declare one or more variables to be **single precision.**

Syntax: DEFSNG *letter*[-*letter*]

All variable names beginning with any of the specified *letters* (case-insensitive) will be single precision, unless a type-identifier character (%, &, #, $, ~, or @) is used to override the DEFSNG declaration.

The DEFSNG declaration must be processed before any of the declared variables are used.

See also: **DEFDBL** **DEFINT** **DEFLNG** **DEFSET** **DEFSTR**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

DELETE **Delete Program Lines** Command

Purpose: Erase a specified range of program *lines*.

Syntax: DELETE [*start*] [-[*end*]]

start Line number or **label** of the first line to delete. If *start* is the only parameter on the line, then only *start* is deleted.

end Line number or **label** of the last line to delete.

If a single line number is specified, the line must exist or an error will be generated. If a range is specified, no error is generated if the lines do not exist.

See **PW@** for the default *start* and *end* values.

DIM **Dimension Array(s)** Statement

Purpose: Declare one or more user **array** variables, and for each, allocate memory space for its specified number of elements.

Syntax: DIM *variable*(*subscripts*) [,*variable*(*subscripts*)] ...

subscripts a list of numbers separated by commas, where each number establishes a dimension for the **array**. There could legally be as many as 255 numbers separated by commas (255 legal dimensions).

The value of each number, limits the maximum subscript for that dimension. (32,767 maximum limit)

e.g. DIM A(2,3) establishes a two dimensional array. Analogous is a "row-column" matrix, with three "rows" 0,1,2 and four "columns" 0,1,2,3. This array would have twelve elements, (3 rows * 4 columns = 12 elements), each addressed by its unique combination of subscripts.

The DIM statement must be processed before the **array** is used.

An array may not be redimensioned, unless it has first been deleted by means of an **ERASE** or **CLEAR** statement.

The first element of a 1 dimensional array is element(0).
The first element of a 2 dimensional array is element(0,0).
The first element of a 3 dimensional array is element(0,0,0). etc.

See also: **Differences** from GWBASIC **Program Initialization**

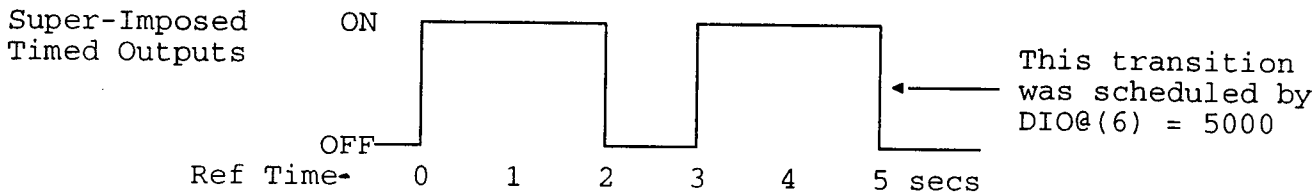
Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

DIO@ **Discrete I/O Point** ORMEC Variable

```
Example 1:  DIO@(6) = ON     'turn discrete output point 6 ON.

Example 2:  DIO@(6) = 50     'turn discrete output point 6 ON
                             'schedule point 6 OFF at (time now + 50 ms)
Example 3:  DIO@(6) = -50    'turn discrete output point 6 OFF
                             'schedule point 6 ON at (time now + 50 ms)

Example 4:  DIO@(6) = 5000   'turn point 6  ON now (ref: time 0)
                             'schedule point 6 OFF at time (0+5) or 5 secs
            WAIT 2000        'wait for 2 secs      (ref: time 0)
            DIO@(6) = -1000  'turn point 6 OFF now (ref: time 2)
                             'schedule point 6  ON at time (2+1) or 3 secs
```



Example 5: IF DIO@(START) THEN GOSUB START
Call a subroutine named START if the Discrete I/O point referenced by the integer variable, START, is ON. Note that while this statement normally would be used to make a decision based on an input point, an output point can also be used.

Note that output points 1 to 40 of the controller are turned OFF by pressing the red RESET pushbutton on the front of the unit or by a **Watchdog Timer** timeout. **Pamux** output points are unaffected.

Note that timed output points will be automatically turned OFF by **errors** which are not **user trapped**. With error trapping enabled, the error handler routine should turn OFF timed outputs as appropriate using **DIO@(point)=OFF**.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

DIST.MOVE@ **Move Distance Command** ORMEC Variable

Purpose: Return the current real-time value of cumulative commanded distance resulting from **MOVE** commands for a servo axis.

Usage: ***var&*** = DIST.MOVE@[(***axes***)]

var& A **long integer** cumulative value of the currently commanded distance resulting from MOVE commands. DIST.MOVE@ is a read-only variable whose range is determined by the **axis position units** conversion factor.

Assigning a value to **MODE@** automatically takes the current position value stored in **POS.ACT@** and assigns that value to **DIST.MOVE@**. The other three distance variables, **DIST.GEAR@**, **DIST.GRAT@**, & **DIST.BIAS@**, are then cleared to zero.

Assigning a value to **POS.ACT@** automatically assigns the same value to **DIST.MOVE@** and clears the other three distance variables to zero. **POS.CMD@** is the sum of DIST.MOVE@, DIST.GEAR@, DIST.GRAT@, & DIST.BIAS@.

DSP.CTR@ **DSP Motion Counter** ORMEC Variable

Purpose: Returns or presets a counter which counts the number of motion statements that have been completed by the DSP.

Syntax: DSP.CTR@(**axes**) = *preset*

Usage: *var%* = DSP.CTR@(**axes**)

preset An integer expression, in the range 0 to 32767, providing the value to which DSP.CTR@(**axes**) is to be preset.

var% An integer variable in which the current value of DSP.CTR@(**axes**) will be placed.

Each time a DSP axis completes execution of a **motion statement**, it increments the value of DSP.CTR@(**axis**) by one. When the value reaches 32767, it will be reset to 0 on completion of the next motion statement. This can be used to track queued and **REPEAT** moves as follows:

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

DUMP **List Symbol Table** Command

Purpose: List to the MotionPRO Development Port, the current state of the symbol table.

Syntax: DUMP

EDIT **Display a Program Line for Editing** Command

Purpose: Enter the **line editor**, display the specified program line. The **edit keys** are then used for editing the line. See also **Text Editor**.

Syntax: EDIT [*line*]

line The line number or **label** of the line to edit. If *line* is missing, then the most recently added line or the line where the last error occurred is used.

An "undefined line number" error occurs if the line number to edit is not in the current program.

The program write enable variable, **PW@**, must be set properly to allow a line to be edited.

EIO.MODE@ **Extended I/O Configuration** ORMEC Variable

Purpose: Define the configuration mode of the Extended I/O Connector.

Syntax: EIO.MODE@ = *mode* Set Discrete I/O Mode

Usage: ***var%*** = EIO.MODE@ Get Extended I/O Mode

mode An **integer** expression representing the Extended I/O Mode.

0 - Digital In this (default) mode, the 24 I/O points at the **Extended I/O Connector** are compatible with 4, 8, 16 or 24-slot Opto-22 racks with the I/O point assignments as shown below.

1 - Pamux In this mode, the Extended I/O connector is used to interface up to 512 digital or analog I/O points using the Opto-22 **Pamux** standard.

var% An integer variable to receive the current value of EIO.MODE@

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

ELS2@ **Electronic Limit Switch Two** ORMEC Variable

Purpose: Return a read-only Boolean variable which indicates whether or not the actual position for an axis is within a specified range. Serve as a gating signal for **BSEN@** of companion axis. (See: **SENS.MODE@**)

Usage: ***var%*** = ELS2@[(***axes***)] 'w/ multiple axes reference, ELS2@s are ORed

var A **Boolean** value which will be turned **ON** or **OFF** at any **servo loop update** for which the value of **POS.ACT@** for an axis moves "into" or "out of" the range specified by **ELS2.LOW@** and **ELS2.HIGH@**.

All three **electronic limit switches** have corresponding TTL-level outputs at the "DSP I/O Header" on the **DSP Axis Module**.

Example:
If ELS2.LOW@ = 90 and ELS2.HIGH@ = 180, then ELS2@ will be **ON** whenever 90 <= **POS.ACT@** < 180. Otherwise it will be **OFF**.

ELS2.LOW@,ELS2.HIGH@ **Electronic Limit Switch Range** ORMEC Variable

Purpose: Define the range for which an **electronic limit switch** will be **ON**.

Syntax: ELS2.LOW@[(***axes***)] = *position1*
ELS2.HIGH@[(***axes***)] = *position2*

Usage: ***var1&*** = ELS2.LOW@[(***axes***)]
var2& = ELS2.HIGH@[(***axes***)]

position1 A **long integer** expression which represents the "lower" of two values which define the range for which a corresponding **electronic limit switch ELS2@** will be ON.

position2 A **long integer** expression which represents the "higher" of two values which define the range for which a corresponding **electronic limit switch ELS2@** will be ON.

Setting *position1* & *position2* to the same non-zero value sets the ELS ON.
Setting *position1* & *position2* to zero sets the ELS OFF.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

END **Terminate Program** Statement

Purpose: Terminate a program and **CLOSE** all **OPEN** devices & files.

Syntax: END

The END statement turns off all error and event trapping, and event checking completely.

Any Input task or Output task that is in process will be terminated without error.

See Also: **STOP** **ON ERROR** **EVENT**

EOF **End-of-File Status** Function

Purpose: Check for end-of-file status on a file or device.

Usage: ***var%*** = EOF({ *filenum* | *devnum* })

filenum The number under which the file was **OPEN**ed.

devnum The number under which the device was **OPEN**ed.

With a file, if an end-of-file has been reached, EOF returns -1 (**TRUE**); otherwise, it returns 0 (**FALSE**).

With a communications device, an EOF of -1 (**TRUE**) means the input buffer is empty.

See Also: **ITM Driver Support** **Sequential Files**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

ERR$ **Error Message** Function

Purpose: Return the error message string for a MotionBASIC error.

Usage: ***var$*** = ERR$(*error number*)

error number An **integer** expression indicating the number of the **error code** of the error whose "error message" is to be returned. This function returns the same string that MotionBASIC prints to the console when an error occurs.

ERR$(ERR) returns the error message for the most recent error to occur.

See also: **ERL** **ERROR** **ON ERROR** **RESUME**

ERROR **Force Error** Statement

Purpose: Produce error number *n*.

Syntax: ERROR *n*

n An integer expression from 1 to 65535.

If *n* is not one of MotionBASIC's standard **error codes**, and no **ON ERROR** routine is in use, ERROR *n* stops program execution and produces an "User Defined Error" error message.

If an ON ERROR routine is in effect, an ERROR *n* statement is trapped in the normal way, whether or not the number is one of MotionBASIC's error codes.

See also: **ERL** **ERR** **ERR$** **ON ERROR** **RESUME**

ESTOP.OK@ **Emergency Stop Input OK** ORMEC Variable

Purpose: Return the status of the **Emergency Stop** Input.

Usage: ***var%*** = ESTOP.OK@

var% A **Boolean integer** variable which will be TRUE (-1) if +24 VDC is applied to the E-Stop input. Otherwise it will be FALSE (0).

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Allowing "nesting" of Event Handlers:
As explained above, nesting of event handler routines is avoided with implicit MotionBASIC instructions. For those users who want to circumvent this "standard" approach, this discussion is included:

Within an event routine, to permit the execution of **other** event routines, execute a global EVENT ON statement. All other events that were viable, could now be serviced. Remember, only one additional handler routine will be serviced at a time, since along with each GOSUB to an event handler routine, another implicit EVENT STOP is executed.

Note: If an event is to be permitted to either interrupt itself, or some nested event handler that is being executed, then the statement **EVENT DIO@(n) ON** will need to be strategically placed in order to remove the individual "stopped status" that was implicitly commanded. Remember, an individual event "stop status" is implicitly imposed on an event, whenever its event handler routine is called.

Error Handler Ramifications:
In the execution of an Error Handler routine (established via an **ON ERROR** statement) an implicit global EVENT STOP is executed. This inhibit to all Event Handler routines is lifted when the Error Handler executes a **RESUME** statement. If a **RESUME** is not used to exit an Error Handler, then restoring Event Handler execution must be done with a global EVENT ON statement and individual "EVENT DIO@(n) ON statements for any points that could have been caught "individually stopped" within their own Event Handler routines.

Direct Mode Ramifications:
Events are normally "OFF" in **direct** mode. If a program enters direct mode from a **STOP** statement or Ctrl-C, the events are "stopped". A subsequent **CONT**, **GOTO**, or **GOSUB** (implied or explicit) command restores event handling to its previous (program execution) state. An **END** statement turns off events.

See Also: **Stack Clear** **MotionBASIC Event Processing**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Purpose: Identify the set of current fault(s) with a motion controller; Clear (or set) this variable, and reset (or trip) the **No Fault Relay**.

Usage: ***var~*** = FAULT@ 'FAULT@ is returned as a **Set**
Syntax: FAULT@ = { ON | OFF } 'FAULT@ is written as a **Boolean**

var~ A **set** variable to receive the set of current fault condition(s).

OFF Clear FAULT@ and energize the No Fault Relay.
ON Set FAULT@ (8 **Machine Fault**) and trip the No Fault Relay.

Code	Fault Condition	Code	Fault Condition
1	**RAM Checksum Error**	6	**E-Stop Input Open**
2	**Battery Failure**	7	**Axis Fault**
3	**Invalid Unit ID**	8	**Machine Fault**
4	**Internal Error**	9	**String Space Fault**
5	**Axis Module Failure**	10	**MotionCARD Fault**

The FAULT@ variable is "latched", in that when one or more of the fault conditions listed above has occurred, the appropriate elements in FAULT@ are asserted and when the condition "goes away" the fault condition(s) is(are) NOT automatically cleared. To clear FAULT@, you must explicitly clear it using one of the syntaxes above.

Whenever any elements of FAULT@ are non-zero, the green "NO FAULT" indicator on the front of the unit will be OFF, and the **No Fault Relay** will be "dropped out" opening its isolated, normally-open contact output. When FAULT@ is cleared, AND **ESTOP.OK@** is **TRUE**, the coil of the **No Fault Relay** will be energized.

Example: If we assume that both **Battery Failure** and **Invalid Unit ID** were detected at powerup, then the program below would print as shown.

```
10 PRINT FAULT@
20 END
RUN
{2,3}
```


Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

FILES **Display Files in Memory Disk** Command

Purpose: Display the names of files currently stored on a **Memory Disk**. Or, transfer files between the Memory Disk and the Development Computer disks directly.

Syntax 1: FILES ["*filemask*"] List files on Memory Disk
Syntax 2: FILES LOAD "*pcfile*" [TO "*mdfile*"] Load file into Memory Disk
Syntax 3: FILES SAVE "*mdfile*" [TO "*pcfile*"] Save file to PC

filemask A string expression that follows **DOS file-naming** conventions. This string can be used to selectively display the names of files stored in the **Memory Disk**. The "*" and "?" characters may be used as wild cards. If the filemask is not specified, the default value is "*.*" which will cause the names of all the files to be listed.

pcfile The file specification for a file on the Host PC. Disk drive and **path** information may be included in the specification. If the Disk Drive and path specification is not specified, the Host PC's current drive and working directory will be assumed.

mdfile The file specification for a file on the **Memory Disk**. The drive specification and path information is optional. If the drive and path information is omitted, the current Memory Disk directory will be assumed.

See also: **CHDIR** **MKDIR** **RMDIR** **KILL** **NAME** **FORMAT**

FIX **Truncate to Whole Number** Function

Purpose: Truncate, without rounding, any numeric expression to a whole number.

Usage: ***var*** = FIX(***numexpr***)

Note: FIX eliminates everything to the right of the decimal point. FIX returns either an **Integer** or a **Long Integer** depending on **numexpr**.

Example:

```
PRINT FIX(35756.998)
35756

PRINT FIX(-234.678)
-234
```

See also: **INT**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

The loop is skipped if *start* is greater than *end* and *increment* is positive, or *start* is less than *end* and *increment* is negative.

Syntax 2: When taking an action for a **set** of elements, processing of the FOR...NEXT block begins by assigning the lowest numbered element in *setexpr* to *setvar*. The statements between the FOR statement and the NEXT statement are then executed. When the NEXT statement is executed, *setvar* is assigned the next lowest numbered element in *setexpr*. The process repeats until the highest numbered element in *setexpr* is used. Execution then passes to the statement following the NEXT statement.

Note that changes made within the FOR...NEXT loop to *setvar* or *setexpr* have no affect on the sequence of elements assigned to *setvar*.

Inclusion of *setvar* in a NEXT statement is optional as long as it is only used to close one FOR statement; if *setvar* is omitted, its value is assumed to be that of *setvar* in the most recent FOR statement.

FOR...NEXT loops may be nested within one another. Each nested FOR must be given a unique *setvar* specification and its corresponding NEXT statement must appear within the enclosing FOR...NEXT block.

Nested loops that have a common termination point may use a single NEXT statement as long as it specifies multiple *setvars* matching the *setvars* in the FOR statements.

The loop is not executed if *setexpr* is an empty set.

MotionBASIC programs should not include **GOTO** statements which branch out of a **FOR...NEXT** loop. See **structured programming** for details.

See also: **WHILE...WEND** **IF...THEN...ELSE** **Differences** from GWBASIC

FORMAT **Format a Memory Disk** Command

Purpose: Format the **Memory Disk** so that it may be used.

Syntax: FORMAT [DELETE]

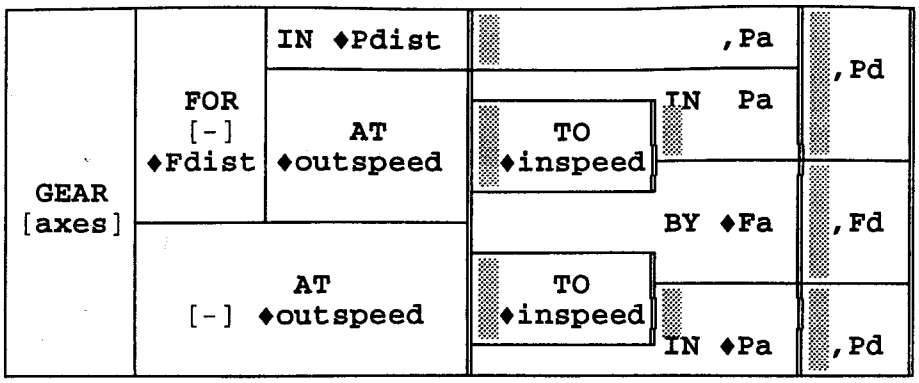
The command will format a **Memory Disk**. If a recognized format exists on the Memory Disk, an error message will be generated to warn the User that a potential exists that data will be lost. To override the error message, the optional DELETE parameter must be specified.

See also: **FILES** **KILL**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

GEAR Statement **SYNTAX CHART**

For Instructions See:
Syntax Chart Rules



♦ indicates that **AXIS.VAR@** can be used for that parameter.

A [**REPEAT**] modifier may precede any **GEAR FOR.**

The following modifiers may be appended to any **GEAR** statement:
[**AFTER** cond]
[**UNTIL** cond]
[**AFTER** cond **UNTIL** cond]

A **GEAR AT** with an [**UNTIL** cond] or both [**AFTER** cond **UNTIL** cond] modifiers is **REPEAT**able

See Also: **GEAR FOR, GEAR AT , EBC**

GEAR **FOR** **Gear Follower Axis a Specified Distance** Statement

Purpose: Move a **follower** motor a relative distance from its current position in synchronism with a **pacer** motor or encoder; move multiple follower motors a relative distance from their current positions.

Syntax:

[**REPEAT**] GEAR [**axes**] FOR **Fdist** IN **Pdist** [,**Pa**[,**Pd**]] [**AFTER** cond] [**UNTIL** cond]

[**REPEAT**] GEAR [**axes**] FOR **Fdist** AT **outspeed** [TO **inspeed**] [IN **Pa**[,**Pd**]]
[**AFTER** cond] [**UNTIL** cond]

[**REPEAT**] GEAR [**axes**] FOR **Fdist** AT **outspeed** [TO **inspeed**] [BY **Fa**[,**Fd**]]
[**AFTER** cond] [**UNTIL** cond]

NOTE: GEAR FOR movements can be "superimposed on a moving frame of reference" created by a constant ratio **GEAR AT** statement.

See Also: **GEAR Syntax Chart Encoder Backup Compensation**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Examples: (assume **user units** of "thousandths of an inch" for axis & pacer)
GEAR {2} AT 11 TO 23 'move axis 2 at 11/23 of the **pacer** rate

GEAR {2} AT 1 TO 1 BY 200,300 **AFTER ASEN@ UNTIL BSEN@**
Cause axis 2 to accelerate to a 1:1 ratio with the pacer within 0.2 inches of its own travel after the A-Sensor is asserted; also begin deceleration when the B-Sensor is asserted and come to rest in 0.3 inches of travel.

REPEAT GEAR {2} AT 1 TO 1 BY 200,300 **AFTER ASEN@ UNTIL BSEN@**
Cause axis 2 to start the motion described above each time the A-Sensor is asserted. (B-Sensor also needs to occur to stop the geared motion.)

Parameters for the GEAR **statement**

Parameter	Description	Default
axes	a **set** of one or more servos to be geared	**AXIS.SET@**
Fdist	total distance & relative direction to gear for	
Pdist	total **pacer** input distance in which to gear for	
outspeed	*(**OUTSPD.MUL@**)= the numerator of the ratio of axis encoder counts to pacer counts	
inspeed	*(**INSPD.MUL@**)= the denominator of the ratio of axis encoder counts to pacer counts	1
,Pa	**pacer** input distance in which to accelerate	Pdist/3
IN Pa	pacer input distance in which to accelerate	0
Pd	pacer input distance in which to decelerate	Pa
Fa	**follower** output distance in which to accelerate	
Fd	follower output distance in which to decelerate	Fa

GEAR Statement parameters are specified in **User Units.**

See Also: **GEAR Syntax Chart** **Encoder Backup Compensation**

Pdist is the total **pacer** input distance in a **GEAR FOR** statement. Along with total pacer distance, the user may specify acceleration distances.

Even when specifying **pacer** acceleration distance parameters (**Pa**, **Pd**) the resultant motion may actually use a Pa=0 and Pd=0 **if Pdist is less than a minimum value**. That minimum value, in **user units**, is given by:

$$\textbf{MIN Pdist} = \frac{\textbf{PCR.SPD.LIM@} * \textbf{PCT.REV@}}{\textbf{LOOP.RATE@}} * 0.14 * \frac{\textbf{PPS.DIV@}}{\textbf{PPS.MUL@}}$$

With the minimum Pdist requirement is satisfied, then a non-zero Pa and non-zero Pd can be performed. **Pa & Pd also have minimum usable values.** These minimum values are:

$$\textbf{MIN Pa} = \textbf{MIN Pd} = \frac{\textbf{MIN Pdist}\ \text{(from above)}}{4}$$

Pa & Pd minimums also apply in a **GEAR AT** statement.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

GET **Read from a Random Access File** Statement

Purpose: Transfer a specified record from a specified random access file on a **Memory Disk** into the associated random access file buffer.

Syntax: GET [#]*devnum* [,*recnumber*]

devnum An **integer** expression specifying a device from 1 to 15.

recnumber A positive, non-zero long expression specifying the record number to be transferred. Defaults to the next record, or record 1 (if no record has been **GET**/**PUT**).

A **FIELD** statement must be declared before execution of a PUT statement.

See Also: **PUT**, **FIELD**

GOSUB **Execute Subroutine** Statement

Purpose: Call a MotionBASIC subroutine.

Syntax: GOSUB {*linenum* | **label**}

linenum A numeric constant representing an existing program line.

label A symbolic **label** at the beginning of an existing program line.

Causes program execution to branch to the specified line number or label; when the **RETURN** statement is encountered, execution branches to the statement immediately following the most recent GOSUB statement or the specified one.

Subroutines may be called any number of times, from any number of different points in a program. They may be nested, and they may have multiple **RETURN** statements.

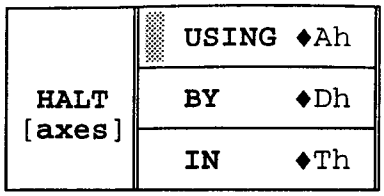
MotionBASIC programs should not include **GOTO** statements which branch out of a subroutine. See the **structured programming** chapter for details.

See also: **GOTO** **ON...GOTO** **ON...GOSUB**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

HALT Statement **SYNTAX CHART**

For Instructions See:
Syntax Chart Rules



The following modifiers may be appended to a HALT statement:

[**AFTER** cond]
[**UNTIL** cond]
[**AFTER** cond **UNTIL** cond]

♦ indicates that **AXIS.VAR@** can be used for that parameter

Parameter	Description	Default
axes	servos to halt	**AXIS.SET@**
Ah	halt rate	**DCL.MAX@**
Dh	halt distance	
Th	halt time	

HALT Statement parameters are specified in **User Units**

HEX$ **Hexadecimal Value, as String** Function

Purpose: Return, as a string, the hexadecimal representation of a number.

Usage: ***var$*** = HEX$(***lngexpr***)

If *lngexpr* is negative, HEX$ returns the twos complement value in hex.

The returned string contains no leading zeros.

See also: **STR$** **OCT$** **BIN$**

HTL.FWD@ **Forward Hardware Travel Limit** ORMEC Variable

Purpose: Read the state of the forward hardware travel limit input latch for a servo or set of servos.

Usage: ***var&*** = HTL.FWD@[(***axes***)] 'w/ multiple axes, HTL.FWD@s are ORed

var A read-only **Boolean** which is latched ON when the forward hardware travel limit input on a **DSP Axis Module** goes "high" (more than 4 volts). If, and only if, the hardware travel limit input is "low" (less than 0.8 volts), HTL.FWD@ may be cleared to the OFF state by the statement **OTL.FWD@** = OFF.

If a servo is in position, velocity or output **mode** and it's forward hardware travel limit is asserted, then:
- it's commanded motion will be stopped using **USR.ACL.LIM@**,
- the **OTL.FWD@** Boolean variable for that axis will be asserted, and
- a **motion error** will be generated.

See **OTL.FWD@** for a detailed explanation of overtravel limit operation.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Syntax 2: IF *expression* THEN *clause* [ELSE *clause*]
Syntax 3: IF *expression* GOTO {*linenum* | **label**} 'not recommended
Syntax 4: IF *expression* THEN *linenum* 'not recommended
Syntax 5: IF *expression* THEN *label* 'not recommended

expression A numeric or **boolean expression.**

clause A MotionBASIC statement or sequence of statements (on the same logical program line, separated by colons).

For syntax 1, if *expression* is non-zero (TRUE), MotionBASIC transfers control to the next program line. It executes the THEN block until the optional matching ELSEIF or ELSE is encountered. The block following the ELSEIF and/or ELSE is skipped. If *expression* is zero (FALSE), MotionBASIC transfers control to the following ELSEIF or ELSE clause at the same level of nesting, if it exists. Unconditional execution continues on the line following the ENDIF. Block IF, THEN, ELSE nesting is allowed.

For syntax 2, if *expression* is non-zero (TRUE), MotionBASIC executes the THEN clause. Otherwise it executes the ELSE clause, if one exists. Unconditional execution continues on the next line.

For syntax 3 & 4, if *expression* is non-zero (TRUE), MotionBASIC branches to (performs a **GOTO**) the line specified by *linenum* or **label**. Otherwise it continues unconditional execution on the following line. **These syntaxes are NOT recommended, but included for compatibility reasons.**

Note that syntax 5 does an implied GOSUB **to the label! This syntax may cause confusion and is NOT recommended.** Instead, use an explicit GOSUB within a block syntax to perform this function.

The THEN and ELSE clauses may also be a line number (with no keyword before it). MotionBASIC treats

IF *expression* THEN {*linenum*}

as if it were

IF *expression* THEN GOTO {*linenum*}

Expression may be a **Boolean** expression (e.g., IF today$ = "Monday" ...) or a numeric expression (e.g. IF A(4) ...). In the latter case, MotionBASIC considers a value of 0 to be FALSE and all other values to be TRUE.

IF statements may be nested to any depth. In nested IFs, all ELSEIFs and ELSEs are associated with their nearest preceding THEN and all ENDIFs are associated with their nearest preceding ELSEIF or ELSE block if there is one otherwise the nearest preceding THEN block.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

INERTIA@ **Total Inertia at Motor Shaft** ORMEC Variable

Purpose: Define the total inertia of a servo axis.

Syntax: INERTIA@[(***axes***)] = *inertia*

Usage: ***var!*** = INERTIA@[(***axes***)]

inertia A **single precision** floating point expression which represents the total system inertia as seen at the motor shaft from 0.000001 to 99 lb-in-sec². Default = 0.0001

Note that when an assignment is made to INERTIA@, an additional range check is made to insure that the ratio of **TRQ.GAIN@** to INERTIA@ is greater than or equal to 12.8. If it is less than 12.8, an error will be generated, and the parameters will not be changed in the DSP.

See Also: **Loop Architecture** **Ka** **Kaf**

INKEY$ **Return a Character from Keyboard** Function

Purpose: Return character(s) from the keyboard buffer (First In/First Out)

Usage: ***var$*** = INKEY$

If no key-press is queued in the keyboard buffer a **null string** ("") is returned. If many keys are pressed, the first in the queue is returned. The string returned will be either **one** or **two** characters in length:

One-Character: **ASCII** character
Two-Characters: CHR$(**0**) + **CHR$**(**scancode**) See Also: **Key Table**

Example:

```
10 TYPICAL.USES:
20   WHILE INKEY$ <> ""   'this small loop flushes out the keyboard buffer
30   WEND                 'by extracting characters from INKEY$ until empty
40 PRINT "Press a key"
50   WHILE INKEY$ =  ""   'this small loop will act as a pause in the code
60   WEND                 'since it will continue until a key is pressed
```


Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

If a prompt is included, it must be followed by either a semicolon or a comma (outside the quotes, before the first variable name). If it's followed by a semicolon, MotionBASIC displays a question mark after the prompt string. If it's followed by a comma, MotionBASIC suppresses the question mark

If no prompt is included, MotionBASIC displays a question mark.

If more than one *variable* is included in the INPUT statement, the user must enter values for all variables, separated by commas.

If only one *variable* is included in the INPUT statement, the user must enter a value for the number and press the Enter key.

If the user enters too few or too many values in response to an INPUT statement, MotionBASIC displays a message telling the user to retry from the start of the input; no values are assigned to variables until satisfactory input is received.

If the user enters a comma in response to an INPUT statement, MotionBASIC displays a "?Redo from Start" error message.

Input for a string variable need not be enclosed in quote marks. String input that is enclosed within quotes is handled as though it were not (that is, MotionBASIC ignores the quotes).

Input to a numeric variable must be numeric.

See also: **INPUT @** **INPUT #**

INPUT # Get Input from a Device or File Statement

Purpose: Receive input from a device or **data file** and assign it to one or more numeric or string variables.

Syntax: INPUT #*devnum*, *variable* [,*variable*] ...

devnum The number under which the device was **OPEN**ed. It may refer to a communications device or data file.

variable The name of a variable (string or numeric) that will receive input.

The input data type must match the variable(s) in the INPUT # statement.

String input need not be enclosed within quote marks, unless it contains one or more line feeds or carriage returns or a significant quote mark or comma.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

INSTR **Find Substring** Function

Purpose: Return the character position within a string at which a substring is found.

Usage: ***var%*** = INSTR([*start*,] *stringexp1*, *stringexp2*)

start An **integer expression** in the range 1 to 255 specifying an optional offset from which to start searching. Default=1.

stringexp1 The string to be scanned.

stringexp2 The substring to be searched for.

INSTR returns 0 if:
start > LEN(*stringexpr2*)
stringexpr1 is null
stringexpr2 cannot be found

INT **Next Lower Whole Number** Function

Purpose: Return the largest whole number "less than or equal to" *numexpr*.

Usage: ***var*** = INT(***numexpr***)

Note: INT rounds to the next lowest whole number. INT returns either an **Integer** or a **Long Integer** depending on **numexpr**. Compare this with **FIX**, which simply eliminates everything to the right of the decimal point.

Examples:

```
PRINT INT(33123.89)
33123

PRINT INT(-12.11)
-13
```


Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

When an INPUT @ statement is executed:

1) The cursor is positioned at the specified row and column. If it was specified, the prompt is printed in standard video.

2) The format field is printed in reverse video, with the current value of *var*. If a variable was specified, it can be edited.

3) The statement is completed when one of the following keys is pressed:

Enter	Exit statement & update variable.	
F1 - F10	Exit statement & update variable.	
↑	Exit statement & update variable.	
←	*Exit statement & update variable.	* These keys first move the internal field cursor to the far left or right.
→	*Exit statement & update variable.	
↓	Exit statement & update variable.	
Esc	Exit statement.	

At completion, **EXITKEY** contains a value for the key used, see **Key Table**.

If the statement was used for a menu selection type field (*var* was omitted), the formatted field is simply re-displayed in standard video, and the only information changed is the value stored in **EXITKEY**.

If the statement changed a variable, the following takes place:

1) If the *Escape* key was pressed, then the variable is not updated, and the variable's last value is displayed in standard video.

2) Otherwise, if no range checking was specified, the variable is updated with the new value and printed in standard video.

3) If a range check was specified, then the variable will be updated with the new value only if it passes the range check. If it fails the range check, a **user trappable error** is generated.

4) A new value in **EXITKEY** indicates the key that terminated the edit.

Example:

```
10 X = 15
20 INPUT @12,10; "Enter Value:  "; FOR 10 to 20; USING " ## "; X
```

The program above will display the following row 12, column 10 on the screen:

```
Enter Value:   15
```

The cursor will be positioned in the left most column of the reverse video field and editing will begin. Editing of the field will continue until one of the **Exit Keys** is entered. After it is entered, a corresponding value will be stored in the **EXITKEY** function, the edited value will be put into the variable X (unless it was the Escape key or the value failed the range check), the reverse video field will be changed to standard, and MotionBASIC execution will continue at the next statement.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

var$ When read, the IO.MODE@ array returns a one character string corresponding to the mode of the specified I/O *point*. Invalid information will cause a "?" to be returned.

The 16 *Integral I/O* each default to I, yet can be set to I, O, R or F.

Example: IO.MODE@(1) = "IIOO ioio FROI RRFF" results in the following:

Point #	I/O Mode	Point #	I/O Mode
1	Input	9	Falling-edge Latch
2	Input	10	Rising-edge Latch
3	Output	11	Output
4	Output	12	Input
5	Input	13	Rising-edge Latch
6	Output	14	Rising-edge Latch
7	Input	15	Falling-edge Latch
8	Output	16	Falling-edge Latch

When EIO.MODE@ = **0** (digital) then I/O points 17 to 40 provided at the **Extended I/O Connector** can be configured as follow:
Points 17-24 are individually configurable as I or O
Points 25-32 are bank configurable as either all I or O
Points 33-40 are bank configurable as either all I or O
Note that setting the IO.MODE@ of any point within a "bank" will change the mode of all the other points in that bank to that setting.

When EIO.MODE@ = **1** (**Pamux**) then I/O points 17 to 528 are configured to align with the addresses of the selected Pamux "Brain-Boards". To calculate the **AIO@** or **DIO@** point number for the first point associated with a brain-board, multiply its Pamux address by 8 and add 17. e.g. The base I/O point on the Pamux brain-board with address 0 maps to I/O point 17, 2 maps to 33, etc. Subsequent points associated with that particular brain-board occupy subsequent point numbers in the array.

Pamux I/O points may be individually configured as I, O, B, C, S or T as long as they are consistent with the **Pamux** brain-board and I/O module

installed in the system. Pamux I/O is unaffected by a **Watchdog Timeout** or the red reset pushbutton on the front of the controller.

Note also that IO.MODE@ is "write only" for Pamux and the configuration cannot be "read back" using the IO.MODE@ variable.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

The THEN and ELSE clauses may also be a line number (with no keyword before it). MotionBASIC treats

IF *expression* THEN {*linenum*}

as if it were

IF *expression* THEN GOTO {*linenum*}

Expression may be a **Boolean** expression (e.g., IF today$ = "Monday" ...) or a numeric expression (e.g. IF A(4) ...). In the latter case, MotionBASIC considers a value of 0 to be FALSE and all other values to be TRUE.

IF statements may be nested to any depth. In nested IFs, all ELSEIFs and ELSEs are associated with their nearest preceding THEN and all ENDIFs are associated with their nearest preceding ELSEIF or ELSE block if there is one otherwise the nearest preceding THEN block.

Note: There are other variations of the IF-THEN-ELSE construct that combine the syntaxes described above, but these variations are not recommended.

See also: **ON...GOSUB** **ON...GOTO** **Operators**

IN.ACL@ **In Acceleration** ORMEC Variable

Purpose: Return a read-only **Boolean** variable as true if the specified servo(s) is (are all) being commanded by a **MOVE** to accelerate.

Usage: ***var%*** = IN.ACL@[(***axes***)]

IN.DCL@ **In Deceleration** ORMEC Variable

Purpose: Return a read-only **Boolean** variable as true if the specified servo(s) is (are all) being commanded by a **MOVE** to decelerate.

Usage: ***var%*** = IN.DCL@[(***axes***)]

IN.MOT@ **In Motion** ORMEC Variable

Purpose: Return a read-only **Boolean** variable as true if the specified servo(s) is (are all) being commanded by a **MOVE** to move.

Usage: ***var%*** = IN.MOT@[(***axes***)]

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

KP@ **Position Loop Gain Factor** ORMEC Variable

Purpose: Define the position loop gain factor for a servo axis.

Syntax: KP@[(***axes***)] = *factor*

Usage: ***var%*** = KP@[(***axes***)]

factor An **integer** expression which represents position loop gain from 1 to 500% for **torque mode servos** (**VEL.GAIN@** = 0), and from 0.1 to 30 in/min/mil for **velocity mode servos** (**VEL.GAIN@** > 0).

Defaults are 100 for **torque mode servos**, and 5 for **velocity mode servos**, respectively.

See Also: **Loop Architecture** **Kp**

KPI@ **Position Loop Integral Gain Factor** ORMEC Variable

Purpose: Define the position loop integral gain factor for a servo axis.

Syntax: KPI@[(***axes***)] = *factor*

Usage: ***var%*** = KPI@[(***axes***)]

factor An **integer** expression which represents the position loop integral gain factor as a percentage from 0 to 500%. Default = 0

KPI@ would normally be changed from the default of 0 to 100% for applications where position error during motion is critical, such as electronic gearing or in the control of continuous web systems.

See Also: **Loop Architecture** **Kpi**

KVF@ **Velocity Feedforward Factor** ORMEC Variable

Purpose: Define the velocity feedforward gain factor for a servo axis.

Syntax: KVF@[(***axes***)] = *factor*

Usage: ***var%*** = KVF@[(***axes***)]

factor An **integer** expression which represents the velocity feedforward gain factor as a percentage from 0 to 200%. Default = 100

See Also: **Loop Architecture** **Kvf**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

LCASE$ **Convert Upper Cases to Lower Cases** Function

Purpose: Return a string with all upper case characters of its argument being converted into lower case characters.

Usage: ***var$*** = LCASE$(*stringexpr*)

stringexpr A string expression.

See also: **UCASE$**

LEFT$ **Substring at Left** Function

Purpose: Return a specified number of characters from the left side of a string.

Usage: ***var$*** = LEFT$(*stringexpr,length*)

stringexpr A string expression.

length An **integer expression** in the range 0 to 255 which determines the number of characters to be returned. If length is larger than **LEN**(*stringexpr*), LEFT$ returns the entire string. If length is 0, LEFT$ returns a null string.

See also: **MID$** **RIGHT$**

LEN **Length of String** Function

Purpose: Return the number of characters in a string, including blanks and nonprinting characters.

Usage: ***var%*** = LEN(*stringexpr*)

stringexpr A string expression.

LET **Assignment** Statement

Purpose: Assign the value of an *expression* to a *variable*.

Syntax: [LET] *variable* = *expression*

variable A valid variable name.

expression The value assigned to *variable*. If *variable* is numeric, then *expression* must be numeric. If *variable* is a string, then *expression* must be a string.

Note: The keyword LET is optional and is rarely used.

See also: **=**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

LOC — Last Accessed Record — Function

Purpose: Return the current position in a data file OR
Return the status of serial communications devices.

Usage: ***var#*** = LOC({ *filenum* | *devnum* })

filenum is the file number used when the file was **OPEN**ed.

devnum is the device number used when the device was **OPEN**ed.

With **Random Access Files**, LOC returns the record number last accessed with a **GET** or **PUT** statement.

With **Sequential Files**, LOC returns the number of 128-byte blocks accessed. For Input files, the first sector is read automatically when the file is opened. LOC will start at 1.

For communication devices, LOC returns the number of characters in the input queue waiting to be read.

LOCATE — Position the Cursor — Statement

Purpose: Set the position and appearance of the cursor.

Syntax: LOCATE {*row*|[*row*],{*col*|[*col*],*cursor*}}

Typical: LOCATE *row* [,*col* [,*cursor*]]

row A numeric expression to set a row position of the cursor. Range is 1 to the screen maximum less any rows used for **function key text**. Defaults to the current row position.

col A numeric expression in the range 1 to **WIDTH**. Sets the column position of the cursor. Defaults to current col.

cursor A numeric expression. If 1, the cursor is visible; if 0, the cursor is invisible. Defaults to a visible cursor.

See also: **CSRLIN** **POS**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

LSET **Left-Justify String Data** Statement

Purpose: Left-justifies string data in a specified field of the random buffer, in preparation for a **PUT** statement or left-justifies string data in ordinary **string variables**.

Syntax: LSET *strvar* = *stringexpr*

strvar A valid string variable name; specifies the variable into which *stringexpr* is to be left justified.

stringexpr A string expression.

If **LEN**(stringexpr) is less than **LEN**(strvar), strvar is padded on the right with space characters.

See Also: **RSET**, **PUT**

MBVER$ **MotionBASIC Version** Function

Purpose: Return the version code for MotionBASIC.

Usage: ***var$*** = MBVER$

var This read-only **string** variable returns a string representing the version of MotionBASIC. e.g. For MotionBASIC version 1.1a, then the string returned will be "1.1a".

Note that in MotionBASIC version code, the number before the decimal point refers to a "major" release which adds significant new features. The second number refers to a "minor" release which adds new features, and possibly introduces some incompatibilities. The alphabetic character refers to a change which adds "compatible new features and fixes" to a minor release.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MID$ **Substring in Middle** Function

Purpose: Return a specified number of characters from a string, beginning at a specified character position.

Usage: ***var$*** = MID$(*stringexpr*, *position* [,*length*])

stringexpr A string expression.

position An **integer expression** in the range 1 to 255; specifies the character position from which the substring is to be extracted. If *position* is greater than **LEN**(*stringexpr*), a null string is returned.

the number of characters to return. If length is omitted, MID$ returns all the characters in the string beginning at character *position*.

See also: **LEFT$** **RIGHT$**

MKDIR **Makes a new directory on the Memory Disk** Command

Purpose: Makes a new directory on the **Memory Disk**.

Syntax: MKDIR "[1:]**path**"

path A string expression that follows DOS path specification conventions which specifies the name of the directory to be created.

See also: **CHDIR** **RMDIR** **FILES**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Feature Provided	OPERATING MODES 0	1	2	3	4	5	DEBUG MODES 13	14	15
Position Feedback	●	●	●	●	●	●	◔	◔	◔
Signal Loss Detection		●	●	●	●	●			
Alarm Detection			●	●	●	●			
Overtravel Limit Detection				●	●	●			
Torque Output Enabled				●	●	●	◔	◔	◔
Velocity Control Loop On					●	●		◔	◔
Position Control Loop On						●			◔

- **In Mode 0**, all **axis fault** and **overtravel limit** checking is disabled.
- **In Mode 1**, the encoder interface is checked for **signal loss**, but servodrive **alarm detection** and overtravel limit checks are disabled.
- **In Mode 2**, the encoder interface is checked for signal loss, AND servodrive alarm detection. Overtravel limit checks are disabled.
- **In Mode 3-5**, all **axis fault** and **overtravel limit** checking is operational.
- **Servo operation in modes 13-15 MAY BE UNSAFE and should NOT be attempted.**

Modes 13-15 are similar to modes 3-5, but with **alarm detection**, encoder **signal loss** detection, and **hardware overtravel limits** disabled. These modes are provided for uses such as program debug with no servodrive connected to the axis module. **Servo operation in modes 13-15 MAY BE UNSAFE and should NOT be attempted with a servodrive and motor connected to the axis controller.**

Mode 15 can also be useful to create a virtual axis, or ideal motor, when a perfect motion generator is desired for sourcing commanded position information onto **MotionDATA** for **electronic gearing** of real servo axes.

When **MD.MODE@** is set at 10 or more, the next change to MODE@ will cause the axis to begin monitoring **MotionDATA**. This is also when the individual DSP Axis Modules' cycle is synchronized to the Axis sending MotionDATA.

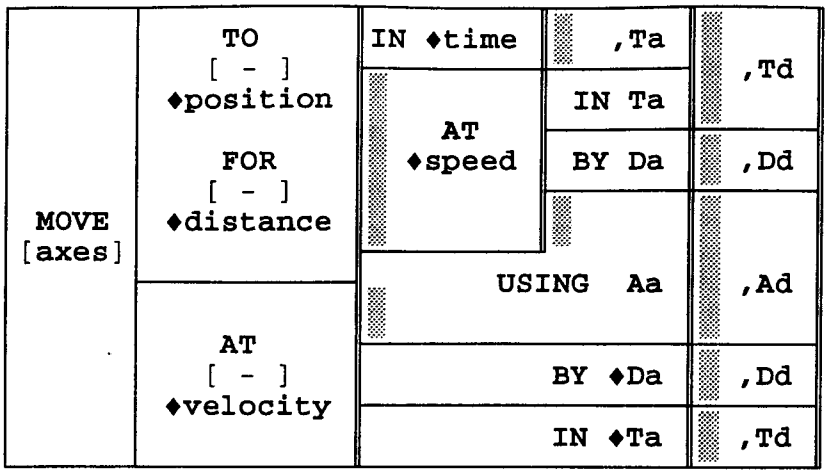
Assigning a value to **MODE@** automatically takes the current position value stored in **POS.ACT@** and assigns that value to **DIST.MOVE@**. The other three distance variables, **DIST.GEAR@**, **DIST.GRAT@**, & **DIST.BIAS@**, are then cleared to zero.

Note: Setting **MODE@** to 5 (or 15) while **PERR.NORM@** is non-zero will produce a torque that is proportional to the sum of PERR.NORM@ and **POS.ERR@**.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MOVE Statement **SYNTAX CHART**

For Instructions See:
Syntax Chart Rules



A [**REPEAT**] modifier may precede a **MOVE FOR**.

The following modifiers may be appended to a MOVE statement:

[**AFTER** cond]
[**UNTIL** cond]
[**AFTER** cond **UNTIL** cond]

♦ indicates that **AXIS.VAR@** can be used for that parameter

See Also: **MOVE FOR**, **MOVE TO**, **MOVE AT**

MOVE **FOR** **Move Servomotor(s) for a Relative Distance** Statement

Purpose: Move a motor a relative distance from its current position. Move multiple motors a relative distance from their current positions.

Syntax:

[**REPEAT**] MOVE [**axes**] FOR **dist** IN **time** [,**Ta**[,**Td**]][**AFTER** cond] [**UNTIL** cond]
[**REPEAT**] MOVE [**axes**] FOR **dist** AT **speed** BY **Da**[,**Dd**] [**AFTER** cond] [**UNTIL** cond]
[**REPEAT**] MOVE [**axes**] FOR **dist** AT **speed** IN **Ta**[,**Td**] [**AFTER** cond] [**UNTIL** cond]
[**REPEAT**] MOVE [**axes**] FOR **dist** [AT **speed**] [USING **Aa**[,**Ad**]] [**AFTER** cond] [**UNTIL** cond]

Note that MOVE FOR statements may be "superimposed on a moving frame of reference" created by a **GEAR AT** statement.

When the [**UNTIL** *cond*] clause is used, the condition is not checked until the commanded motion is within the decel distance (**Dd**) to completion. The minimum move in this case is the **dist** specified.

See Also: **MOVE Syntax Chart**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MOVE **TO** **Move Servomotor(s) to an Absolute Position** Statement

Purpose: Move one or more motor(s) to a specified absolute position.

Syntax:

```
MOVE [axes] TO position  IN time          [,Ta[,Td]] [AFTER cond] [UNTIL cond]
MOVE [axes] TO position [AT speed] [USING Aa[,Ad]] [AFTER cond] [UNTIL cond]
MOVE [axes] TO position  AT speed      BY Da[,Dd]  [AFTER cond] [UNTIL cond]
MOVE [axes] TO position  AT speed      IN Ta[,Td]  [AFTER cond] [UNTIL cond]
```

MOVE TO statements may be "superimposed on a moving frame of reference" created by a **GEAR AT** statement. Only position commanded by MOVE statements is accumulated as absolute position. See **DIST.MOVE@**.

When the [**UNTIL** *cond*] clause is used, the condition is not checked until the commanded motion is within the decel distance (**Dd**) to completion. The minimum move in this case is to **position** specified.

See Also: **MOVE Syntax Chart**

Examples: (assume **user units** of degrees, RPM & msec)

```
MOVE TO -90 IN 100 'move to the -90 degree position in 100 msec

MOVE TO 90         'move to 90 degrees at max speed, accel & decel
```

Examples: (assume **user units** of "thousandths of an inch", "inches per second", acceleration in "g's", and msec)

```
MOVE TO 0
```

Causes the servos defined in **AXIS.SET@** to move to their respective "zero" positions.

Note that these statements will cause the servos to move to the specified positions as defined by the **DIST.MOVE@** variable, which only includes position accumulated from **MOVE statements**.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Parameters for the MOVE statement

Parameter	Description	Default
axes	a **set** of one or more servos to move.	**AXIS.SET@**
position	the end position to reach in a **MOVE TO**.	
distance	the distance & direction in a **MOVE FOR**.	
velocity	the speed and direction in a **MOVE AT**.	
speed	the top speed of a **MOVE FOR** or **MOVE TO**.	**SPD.MAX@**
time	the total time to complete the motion.	
,Ta	time to accelerate.	time/3
,Td	time to decelerate.	Ta
Aa	rate to use for acceleration.	**ACL.MAX@**
Aa,Ad	rates to use for both accel & decel	**ACL.MAX@,DCL.MAX@**
,Ad	rate to use for deceleration.	Aa
Da	distance to move during acceleration.	
,Dd	distance to move during deceleration.	Da

MOVE Statement parameters are specified in **User Units**. See Also: **MOVE Chart**

Generation III Configuration Subroutine

MP.CONFIG is included in the list of **keywords**, but it's not a keyword in the strictest sense. It is the **label** of the configuration subroutine that is automatically generated & downloaded by **MotionPRO**.

The purpose of this subroutine is to setup the motion controller for a particular machine configuration. This is done by initializing **ORMEC variables** which are dependent on: the axis motor selection, the user units selected, operating ranges chosen for the machine, specific servo tuning parameters, operating variables and I/O configuration. **It must be executed each time that: 1)** power is applied to the unit, **2)** a **RUN** command is executed, **3)** a **CLEAR** statement is executed. It initializes a number of variables stored in the volatile memory of all the **DSP Axis Module**(s) in the system.

MP.CONFIG also supports the polling & enabling of **absolute encoders**.

Use of this subroutine name as a **label** elsewhere in a user program is not recommended as it will disable this valuable feature!

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

NAME **Change the Name of a Memory Disk File** Command

Purpose: Change the name of a file or directory that had previously been saved or created on the **Memory Disk**.

Syntax: NAME **cardfile** AS "**filename**[**.ext**]"

cardfile "[1:][*path*\]*filename*[*.ext*]"

path A string expression that follows **DOS file-naming**
filename conventions.
ext

filename A string expression that follows **DOS file-naming**
ext conventions.

See also: **FILES** **KILL**

NEW **Delete Current Program** Command

Purpose: Delete the program currently in memory, clear all variables, and set trace mode off.

Syntax: NEW

The *extent* of influence of the NEW command is dependent on the value of the program write enable variable **PW@**. Attempting a NEW command with **PW@** = 0 will cause an error.

NEW RESTORE **Restore Non-Volatile Memory** Command

Purpose: Restore all non-volatile memory to factory default values and save the current state of the Gen-III unit.

Syntax: NEW RESTORE

This command will restore all non-volatile memory to factory default values, save the current state of the Gen-III unit to a **disk** file through the serial link to **MotionPRO**, and reset the Gen-III unit. It should be used only when non-volatile memory is suspected of being corrupted. Please call **ORMEC Service** to report its use.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Purpose: Enable the trapping of error conditions and specify the first line of an error-trapping routine. Also, disable error trapping.

Syntax 1: ON ERROR GOTO {*linenum* | **label**} Enable error trapping
Syntax 2: ON ERROR GOTO 0 Disable error trapping

linenum The first line of the error-handling routine. Setting *linenum* to 0 disables error trapping, but does not transfer program control to line 0.

label A symbolic **label** at the beginning of the error-handling routine.

To use ON ERROR to trap error conditions, the user must write an error-handler routine which performs appropriate error processing based on the **error codes** returned by the **ERR** function, the **ERR$** function, and the **ERL** function, as well as various fault variables such as: **FAULT@**, **AXIS.FAULT@**, **AXIS.FLT1@**, **AFAULT@** and **ALARM@**.

Within an Error Handler, before a **RESUME** statement is executed, error trapping (and all other **Event processing**) is disabled. If an error occurs during this time, MotionBASIC halts program execution, returns to **direct** mode and displays the error message. Error handlers must contain a **RESUME** statement.

The statement "ON ERROR GOTO 0" has special meaning. When used in a program, it disables any further error trapping. When used within an error-handler routine, it causes MotionBASIC to immediately process the trapped error as it would without an error-handler. Error-handlers should use an "ON ERROR GOTO 0" statement to process errors not to be specifically processed within the error-handler routine.

Without an error-handler enabled, MotionBASIC normally prints the error code (**ERR**), error message (**ERR$**(ERR)), and the line number where the error occurred (**ERL**) to the MotionPRO port, and then returns to direct mode. For some errors, such as **syntax errors**, it opens the **line editor** with the cursor at the line location where the error was found.

An example program flow for an Error Handler:

```
ERR.HDLR:                                  'Previously defined in ON ERROR statement
'  user code                               'Here program is vulnerable to an error
'  user code
RESUME MORE.ERROR.HDLR                     'Error trapping is re-established
                                           'and execution directed to additional
MORE.ERROR.HDLR:                           'code to rectify the current error
'Typically used functions &
'variables in error handlers
'include:
'  ERR    ERL    ERR$
'  FAULT@     AFAULT@
'  ALARM@  AXIS.FLT1@
'  AXIS.FAULT@
STACK CLEAR                                'performed since a restart will be done
GOTO RESTART.MY.MACHINE                    'send execution to a restart routine
```


Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

ON KEY...GOSUB **Define a Key Event Handler Routine** Statement

NOTE: The execution order of ON KEY...GOSUB statements, sets the priority of the key event; First defined has highest priority, etc.

With trapping in effect, MotionBASIC checks for the specified keypress before executing each program statement. When the keypress is detected, MotionBASIC branches to the specified subroutine and temporarily suspends trapping (by means of an implicit **KEY (n)** STOP statement). Trapping automatically resumes when a RETURN from the handling subroutine is executed (unless the handler explicitly performs a KEY (n) OFF or KEY (n) STOP).

Event Keys **do not** enter the keyboard buffer (**INKEY$**)

However in MotionBASIC version 2.1i and later, Event Keys **will** enter the Keyboard buffer if the Trapping of the key is in the OFF state. At that time the key-press is treated as normal input.

See Also: **MotionBASIC Event Processing** **ON EVENT...GOSUB**

ON...GOSUB **Branch to Item in Subroutine List** Statement

Purpose: Cause MotionBASIC to execute one of a list of subroutines.

Syntax: ON *y* **GOSUB** {*linenum* | **label**} [,{*linenum* | *label*}]...

y An **integer** expression, valid range 0 to 255. Upon execution of the ON...GOSUB statement, MotionBASIC branches to the *y*'th item in the list of labels and/or line numbers that follows the keyword GOSUB.
If *y* is 0 or > "# of list items", then execution transfers to the next statement.
If *y* is outside the valid range, an error is generated.

linenum The first line number of a subroutine.

label A symbolic **label** at the beginning of a subroutine.

See also: **ON...GOTO**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

OPEN OIT **Allow Operator Interface Terminal I/O** Statement

Purpose: Make an Ormec **OIT** available for input and/or output.

Usage 1: OPEN "O",[#]0,"SCRN:" 'Redirect output to Video Adapter
Usage 2: OPEN "I",[#]0,"KYBD:" 'Redirect input to Keyboard Interface

Operator I/O for MotionBASIC programs defaults to the MotionPRO Development Port (RS-232). To use an OIT, which is essentially two devices, a remote monitor & a remote keyboard, redirect the interfaces by opening KYBD: and SCRN: both as device number 0.

Assigning device #0 to an OIT allows the use of **PRINT**, **LOCATE**, **INPUT**, and **INPUT @** statements.

No error is generated when "KYBD:" is opened in **direct** mode. However Device #0 is the only valid number.

See also: **CLOSE**

OPEN MMI **Allow MMI-840 I/O** Statement

Purpose: Make an Ormec **MMI-840** available for input and/or output.

Syntax: OPEN "*mode*", [#]*devnum*,"*device*[*serialspec*]MMI"

Typical: OPEN "R",0,"SRL2:MMI" 'open MMI for use on the MotionNet port

mode A **string** expression that defines the MMI operating mode as Input and/or Output with one of the following values:

I	Input Only
O	Output Only
R	Input & Output

devnum An integer expression; sets an MMI device number (0 to 15) Device #0 allows the use of **PRINT**, **LOCATE**, **INPUT**, and **INPUT @** statements on an MMI-840.

Devices #'s 1-15 are accessed with **PRINT #** and **INPUT #.**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Purpose: Make an Ormec **ITM-270** available for input and/or output.

NOTE: When a port/device is opened with ITM support, continuous polling of the ITM-270 takes place. When the port is closed, the polling is stopped. Other statements are effected, see: **ITM Driver Support**

Syntax: OPEN "*mode*",[#]*devnum*,"*device*[*serialspec*]ITM=*addr*[,*time*[,*timeout*]]"

Typical: OPEN "R",1,"SRL2:ITM=1" 'open ITM for use on the MotionNET port

mode A **string** expression that defines the ITM operating mode as Input and/or Output with one of the following values:

I	Input Only
O	Output Only
R	Input & Output

devnum An integer expression; sets an ITM device number (1 to 15) Device #'s 1-15 are accessed with **PRINT #** and **INPUT #**.

device A string expression to define the ITM-270 interface:

		Default Spec
SRL2:	MotionNET (RS-485) serial port	(9600,N,8,1)
COM1:	IBM PC compatible serial port	(9600,N,8,1)
COM2:	IBM PC compatible serial port	(9600,N,8,1)
COM3:	IBM PC compatible serial port	(9600,N,8,1)
COM4:	IBM PC compatible serial port	(9600,N,8,1)

Note: ITM-270 default parameters are (9600,N,8,1)

serialspec A string expression for setting non-default **serial port** communication parameters. Four parameters would be specified, separated by commas:

baudrate	1200, 2400, 4800, 9600
parity	N, O, E (None, Odd, or Even)
databits	7 or 8
stopbits	1 or 2

addr A string constant representing the ITM-270's hardware address setting. Range: 1-32

time A string constant setting the ITM-270's refresh rate. Range: 100 to 5000 msecs. Default is 100 msecs.

timeout A string constant setting the number of refresh cycles that will be permitted without receiving a response from the ITM before MotionBASIC decides the device has failed. Range: 1-10. Default is 1.

Communication parameters at the ITM-270 are changed in its setup mode. This mode is entered by pressing the . key through its powerup cycle.

Examples:
```
OPEN "R",1,"SRL2:ITM=1"                          'simplest form
OPEN "R",2,"COM1:4800,N,8,1,ITM=15,500,2" 'total specification
```

See also: **CLOSE**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

OPEN File **Allow File I/O** Statement

Purpose: Make a **Data File** available for input and/or output.

Syntax: OPEN "*mode*", [#]*filenum*, "*cardfile*"[,*reclength*]

mode A **string** expression that defines the data file I/O mode with one of the following values:

I	Input from a **Sequential File**
O	Output to a Sequential File (Overwrite)
A	Append to a Sequential File (add data)
R	**Random Access File** Input & Output

filenum An integer expression. Specify a file from 1 to 15

cardfile A string expression. Follow **DOS file-naming conventions**

reclength An integer expression, range 1 to 32767. Set the record length for a **Random Access File**. Default is 128.

OR **Logical OR** Operator

Purpose: Perform a logical OR on two binary integers or long integers.

Usage: *exp1* OR *exp2*

exp1, exp2 Any expressions, including **boolean** expressions, that produce an integer or long integer result.

The operands must be either **integer** or **long integer**. Any other operand types will cause a Type Mismatch error to occur. The result is the same type as the largest operand. If the two operands are different sizes, **integer type conversion** is performed.

MotionBASIC performs a **bitwise** OR, according to the OR **truth table**. See the chapter on **bitwise operations** for an example.

See also: **NOT** **AND** **XOR** **Precedence**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Purpose: Identify a reverse overtravel limit condition, or re-enable normal operation after an overtravel condition has occurred.

Usage: ***var%*** = OTL.REV@[(***axes***)] 'w/ multiple axes, OTL.REV@s are ORed

Syntax: OTL.REV@[(***axes***)] = 0

var% a "latched" **Boolean** variable which will be ON if any of the servo axes defined in ***axes*** have exceeded their reverse hardware or software overtravel limits. OTL.REV@ must be turned OFF before any motion commands will again be allowed for that axis.

When a servo's **reverse hardware travel limit** input is asserted or its **actual position** is more negative than the **reverse software travel limit**:
- it's commanded motion will be stopped using **USR.ACL.LIM@**,
- the OTL.REV@ Boolean variable for that axis will be asserted, and
- a **motion error** will be generated which is **user trappable**.

The statement **OTL.REV@ = OFF** will clear reverse *hardware* overtravel limits only if the reverse overtravel limit input on the **DSP Axis Module** is low (less than 0.8 volts). Therefore, to "back off" a hardware limit switch under power, the overtravel limit switch must be overridden, allowing OTL.REV@ to be cleared and the axis to be moved.

The statement **OTL.REV@ = OFF** will clear reverse *software* overtravel limits even though the servo's **actual position** is more negative than than the **reverse software travel limit**. In this case, forward motion commands will be allowed but attempting to move in the reverse direction will set OTL.REV@ and cause the **motion error** to recur. Normal operation will be fully restored once the servo is out of the overtravel zone.

This makes it possible to:
- hit a software limit which stops the motor slightly into the overtravel zone,
- reset OTL.REV@ to zero, even though still in the overtravel zone, and
- then **MOVE** the motor out of the overtravel zone under power.

Please note that for errors which occur asynchronous to program operation, such as **Axis Faults** and Overtravel Limits or **E-Stop**, only the first occurrence will report a **user trappable error**. Additional reporting of these errors must be re-enabled with a FAULT@=0 statement.

See Also: **OTL.FWD@**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

PB1@ **Pushbutton PB1 Pressed** ORMEC Variable

Purpose: Return the status of the black pushbutton labeled "PB1" on the front of the unit.

Usage: ***var%*** = PB1@

var% A **Boolean integer** variable which will be ON (-1) if the black pushbutton labeled "PB1" on the front of the controller is pressed. Otherwise it will be OFF (0).

Note that this pushbutton is also used to:

1) Inhibit a MotionBASIC program from automatically executing at line number 0 on **powerup**.

2) To load a MotionBASIC program named **PB1LOAD.BAS**, stored on a **Memory Disk**.

Pushbutton Load Program File

PB1LOAD.BAS is included in the list of **keywords**, but it is not a keyword in the strictest sense. It is the **required** filename for the program file which is designated to load into the controller from the **Memory Disk** when PB1 button is held pressed during **powerup**.

This program file must be stored in the root directory of the **Memory Disk** in order to provide this functionality.

See Also: **AUTOLOAD.BAS**

PCR.SPD.LIM@ **Pacer Encoder Speed Limit in RPM** ORMEC Variable

Purpose: Define the absolute machine speed limit for a axis **pacer** encoder in RPM of the encoder.

Syntax: PCR.SPD.LIM@[(***axes***)] = *speed*

Usage: ***var%*** = PCR.SPD.LIM@[(***axes***)]

speed An **integer** expression representing the absolute maximum machine speed limit for an axis **pacer** encoder in RPM of the encoder. Default PCR.SPD.LIM@ = 4000 RPM.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

PCT.REV@ **Pacer Counts/Rev** ORMEC Variable

Purpose: Define the number of counts per revolution of the servo **pacer** position transducer.

Syntax: PCT.REV@[(***axes***)] = *counts*

Usage: ***var%*** = PCT.REV@[(***axes***)]

counts An integer expression which represents the number of counts per revolution of the pacer encoder used by a servo axis. Default PCT.REV@ = 6000.

var% A numeric variable to receive the current number of counts per revolution of the pacer encoder as defined by PCT.REV@.

This parameter should be set prior to setting other **pacer user units.**

MAC-E Series Servomotors:	6,000 counts per rev
MAC-A Series Servomotors:	6,000 counts per rev
MAC-B Series Servomotors:	24,000 counts per rev
MAC-C Series Servomotors:	24,000 counts per rev
MAC-F Series Servomotors	
with F-Series "/A" drives:	16,384 counts per rev
with F-Series "/B" drives:	4,096 counts per rev

PEEK **Value at Specified Address** Function

Purpose: Return the byte at a specified memory offset within the current **segment**.

Usage: ***var%*** = PEEK(*address*)

address A numeric expression in the range 0 to &HFFFF, which specifies a memory offset within the **segment** specified by the most recent **DEF SEG** statement.

Accessible memory areas are offsets &H0000-&HFFFF within **segments** &HA000, &HB000, &HC000 or &HD000, except:

- segment &HB000, offsets &H8000-&HFFFF if using a mono monitor, or
- segment &HB000, offsets &H0000-&H7FFF if using a color monitor

See also: **DEF SEG** **PEEKW** **POKE** **POKEW**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

PEEKW **Value at Specified Address** Function

Purpose: Return the word at a specified memory offset within the current **segment**.

Usage: ***var%*** = PEEKW(*address*)

address A numeric expression in the range 0 to &HFFFF, which specifies a memory offset within the **segment** specified by the most recent **DEF SEG** statement.

Accessible memory areas are offsets &H0000-&HFFFF within **segments** &HA000, &HB000, &HC000 or &HD000, except:
- segment &HB000, offsets &H8000-&HFFFF if using a mono monitor, or
- segment &HB000, offsets &H0000-&H7FFF if using a color monitor

See also: **DEF SEG** **PEEK** **POKE** **POKEW**

PERR.INPOS@ **In-Position Error** ORMEC Variable

Purpose: Define the magnitude of position error allowed while the servo axis is still considered to be "in-position".

Syntax: PERR.INPOS@[(***axes***)] = *error*

Usage: ***var&*** = PERR.INPOS@[(***axes***)]

error An integer expression representing the magnitude of position error allowed while the servo axis is considered to be "in-position". This parameter is expressed in **user units**. The range is restricted so that the resulting position error is less than 32,767 counts of the axis position encoder. A value of 0 (default) prevents **IN.POS@** from returning **TRUE**.

When the position error is less than the magnitude specified by this variable, the servo is considered to be "in-position", and the **IN.POS@** Boolean variable will be true.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

PERR.MAX@ **Maximum Position Error** ORMEC Variable

Purpose: Define the maximum allowable position error for the axis servo when it is in a torque producing **mode**.

Syntax: PERR.MAX@[(***axes***)] = *error*

Usage: ***var%*** = PERR.MAX@[(***axes***)]

error An **integer** expression representing the maximum allowable position error for the axis servo in **user units**. The range of this variable is restricted so that the resulting position error in motor encoder counts is less than 32,767. A value of 0 disables maximum position error checking. Default = 360

If this error is exceeded:
- the servodrive output will be disabled,
- **AXIS.FAULT@** will indicate the axis which has a fault, and
- **AFAULT@** will identify the fault.

PERR.NORM@ **Normalization Error** ORMEC Variable

Purpose: Return the current **normalization error**.

Usage: ***var%*** = PERR.NORM@[(***axes***)]

This variable is in **user units**.

When a value is assigned to **POS.ACT@** while **MODE@**=5 (or 15), the current position error, **POS.ERR@**, is added to PERR.NORM@, and the value of POS.ERR@ is set to zero.

If a value is assigned to POS.ACT@ while the axis is in any other Mode, the value of PERR.NORM@ and POS.ERR@ are both set to zero.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

PG.LEN@ **Development Computer Page Length** ORMEC Variable

Purpose: Cause the **LIST** command to pause after printing a certain number of lines.

Syntax: PG.LEN@ = *lines*

Usage: ***var%*** = PG.LEN@

lines An **integer expression** which defines the number of lines to be listed between "pauses". Default = 20. If PG.LEN@ = 0, then there will be no pauses.

Pressing any character (except Ctrl-C & Esc) during a "pause" causes list printing to resume.

Pressing Esc stops the current operation without **Ctrl C** processing.

POKE **Write Byte to Memory** Statement

Purpose: Write a specified byte at a specified memory offset within the current **segment.**

Syntax: POKE *address,byte*

address A numeric expression in the range 0 to &HFFFF, which specifies a memory offset within the **segment** specified by the most recent **DEF SEG** statement.

Accessible memory areas are offsets &H0000-&HFFFF within **segments** &HA000, &HB000, &HC000 or &HD000, except:
- segment &HB000, offsets &H8000-&HFFFF if using a mono monitor, or
- segment &HB000, offsets &H0000-&H7FFF if using a color monitor

byte A numeric expression in the range 0 to &HFF, which specifies the data to be written at offset *address*.

See also: **DEF SEG** **PEEK** **PEEKW** **POKEW**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

POKEW **Write Word to Memory** Statement

Purpose: Write a specified word at a specified memory offset within the current **segment**.

Syntax: POKEW *address,word*

address A numeric expression in the range 0 to &HFFFF, which specifies a memory offset within the **segment** specified by the most recent **DEF SEG** statement.

Accessible memory areas are offsets &H0000-&HFFFF within **segments** &HA000, &HB000, &HC000 or &HD000, except:
- segment &HB000, offsets &H8000-&HFFFF if using a mono monitor, or
- segment &HB000, offsets &H0000-&H7FFF if using a color monitor

word A numeric expression in the range 0 to &HFFFF, which specifies the data to be written at offset *address*.

See also: **DEF SEG** **PEEK** **PEEKW** **POKE**

POS **Current Cursor Column Position** Function

Purpose: Return the column position of the current cursor location.

Usage: ***var%*** = POS(*dummy*)

dummy A dummy argument. Any numeric or string expression may be used.

See also: **CSRLIN** **LOCATE**

POS.ABS@ **Absolute Encoder Poll** ORMEC Variable

Purpose: A read-only variable that when referenced, forces a poll of an absolute encoder, and then returns the net number of encoder counts travelled since the encoder was last physically cleared to zero. It also **enables encoder feedback**, and turns off backup battery drain.

Usage: ***var&*** = POS.ABS@[(***axes***)]

var a **long integer** variable to receive a count value from a poll of an absolute encoder. The count value range is ± ((50000***CNT.REV@**)-1), since POS.ABS@ "wraps around" at ± 50,000 revolutions. POS.ABS@ increments for **counter-clockwise travel**, and decrements for **clockwise travel** of the motor, (regardless of the **CW.FWD@** setting).

The encoder polling task completely "ties-up" a **DSP Axis Module** and takes approximately 1/2 second per axis. All axes interfaced to a **DSP Axis Module** being polled, must have their operating mode disabled, **MODE@**=0, & MotionDATA operating mode set to default, **MD.MODE@**= 0.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

The **required** use of POS.ABS@ for all absolute encoders, should be restricted to the powerup and initialization routines of a program. In the execution of the configuration routine, **MP.CONFIG**, absolute encoders are polled, and the position data received is written to **POS.ACT@**. To insure the use of maximum position resolution (counts), the sequence of statements below, are executed before user units are established.

1) **MODE@**(**AXIS.LIST@**)=0 (all motors are disabled)
2) **MD.MODE@**(**AXIS.LIST@**)=0 (all pacer inputs are disabled)
3) **CNT.REV@** is set to the motor count/rev specification.
4) **POS.MUL@**=1: **POS.DIV@**=1: **POS.MOD@**=0 '(position units: counts)
5) **POS.ACT@**=POS.ABS@ (poll & enable encoder. Set position in counts)
6) **POS.MUL@**, **POS.DIV@**, & **POS.MOD@** are setup to report user units so now **POS.ACT@** is in user units.

Absolute encoders have capacitors that retain enough charge for 4 days of tracking. An optional battery, charges encoders while quadrature feedback is disabled. POS.ABS@ is zeroed only by a complete capacitor discharge.

POS.ACT@ **Actual Position** ORMEC Variable

Purpose: Return the value of, or assign a value to, the actual real-time position of an **axis encoder**.

Syntax: POS.ACT@[(***axes***)] = *position*
Usage: ***var&*** = POS.ACT@[(***axes***)]

position A **long integer** expression which represents the new, real-time position assignment for a servo axis in **axis position units**. The valid range for POS.ACT@ is determined by the axis position units conversion factors and the value of **POS.MOD@**.

var a long integer value equal to the current real-time position of the servo axis in user units

Assigning a value to **MODE@** automatically takes the current position value stored in **POS.ACT@** and assigns that value to **DIST.MOVE@**. The other three distance variables, **DIST.GEAR@**, **DIST.GRAT@**, & **DIST.BIAS@**, are then cleared to zero.

Assigning a value to POS.ACT@ automatically assigns that value to **DIST.MOVE@** and zeroes out: **DIST.GEAR@**, **DIST.GRAT@**, and **DIST.BIAS@**.

If **MODE@**=5 (or 15) when a value is assigned to POS.ACT@, the current position error, **POS.ERR@**, is added to **PERR.NORM@**, and the value of POS.ERR@ is set to zero.

When a value is assigned to **POS.ACT@** while in any other Mode, the value of PERR.NORM@ and POS.ERR@ are both set to zero.

See also: **POS.CMD@** **Normalization Error**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

POS.ASEN@ **Position at A-Sensor** ORMEC Variable

Purpose: Return a value (in **user units**) equal to the real-time position of the axis encoder when the **ASEN@** variable was last asserted.

Usage: ***var***& = POS.ASEN@[(***axes***)]

var A **long integer** read-only variable which indicates the **actual position** of the **axis encoder** when the A-Sensor input last asserted the **ASEN@** variable.

The operation of this feature requires a latched sensor configuration. **ASEN@** must be configured as a **R**ising or **F**alling Latched Sensor with the **SENS.MODE@** variable.

The value of POS.ASEN@ is latched within one microsecond of receipt of the sensor. If however **ASEN@** is asserted within 20 microseconds of **BSEN@**, **DELAY@** or **ZREF@**, the value of POS.ASEN@ will be equal to the value of **POS.BSEN@**, **POS.DELAY@** or **POS.ZREF@** respectively.

POS.BSEN@ **Position at B-Sensor** ORMEC Variable

Purpose: Return a value (in **user units**) equal to the real-time position of the axis encoder when the **BSEN@** variable was last asserted.

Usage: ***var***& = POS.BSEN@[(***axes***)]

var A **long integer** read-only variable which indicates the **actual position** of the **axis encoder** when the B-Sensor input last asserted the **BSEN@** variable.

The operation of this feature requires a latched sensor configuration. **BSEN@** must be configured as a **R**ising or **F**alling Latched Sensor with the **SENS.MODE@** variable.

The value of POS.BSEN@ is latched within one microsecond of receipt of the sensor. If however **BSEN@** is asserted within 20 microseconds of **ASEN@**, **DELAY@** or **ZREF@**, the value of POS.BSEN@ will be equal to the value of **POS.ASEN@**, **POS.DELAY@** or **POS.ZREF@** respectively.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Purpose: Return the current real-time value of commanded position for a servo axis in **user units.**

Usage: ***var***& = POS.CMD@[(***axes***)]

var A **long integer** value which is the currently commanded position. POS.CMD@ is a read-only variable whose range is determined by the **axis position units** conversion factor and the value of **POS.MOD@.**

Commanded position is made up of the sum of four components:

DIST.MOVE@	distance commanded by **MOVE FOR** statements
DIST.GEAR@	distance commanded by **GEAR FOR** statements
DIST.BIAS@	distance commanded by setting this variable directly
DIST.GRAT@	distance commanded by the constant ratio portion of **GEAR AT** statements

The approach taken above allows commanded position generated by **MOVE** or **GEAR FOR** statements to be considered independent of commanded position generated by setting **DIST.BIAS@** or by the execution of continuous ratio **GEAR AT** statements.

An example of the usefulness of this approach is an application where controlled motion is to be superimposed on a moving frame of reference such as a conveyor. If a continuous ratio **GEAR** AT statement is used to create the moving frame of reference, and **MOVE** or **GEAR** statements are used to create the controlled motion, then the separation of sources for the final POS.CMD@ described above allows the user to easily separate the "current position with respect to the conveyor" from the "change of position related to the change in conveyor position".

See also: **POS.MOD@** **POS.ACT@** **POS.ERR@** **PERR.NORM@**

POS.DELAY@ **Position at Delayed Sensor** ORMEC Variable

Purpose: Return a value (in **user units**) equal to the real-time position of the axis encoder when the **DELAY@** variable was last asserted.

Usage: ***var***& = POS.DELAY@[(***axes***)]

var A **long integer** read-only variable which indicates the **actual position** of the **axis encoder** when **DELAY@** was asserted.

The value of POS.DELAY@ is latched within one microsecond of assertion of the DELAY sensor. If however **DELAY@** is asserted within 20 microseconds of **ASEN@**, **BSEN@** or **ZREF@**, the value of POS.DELAY@ will equal the value of **POS.ASEN@**, **POS.BSEN@** or **POS.ZREF@** respectively.

The sensor delay counter is an extra cost option of the DSP Axis Module, **and also requires the use of the appropriate** MotionCARD. **POS.DELAY@ is always 0 without the option and the supporting MotionCARD.**

See Also: **SENS.MODE@**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

POS.ERR@ **Current Position Error** ORMEC Variable

Purpose: Return the current real-time position error of the servo axis.

Usage: ***var&*** = POS.ERR@[(***axes***)]

This error is signed because it is the difference **POS.CMD@**-**POS.ACT@**.

Note that if the **actual position** lags the **commanded position** during motion, POS.ERR@ will be positive for moves in the "forward" direction and negative for moves in the "reverse" direction. Conversely, if POS.ERR@ is negative during a forward move, then the actual position is actually "ahead of" the commanded position.

If a value is assigned to POS.ACT@ when **MODE@**=5 (or 15), then POS.ERR@ is added to **PERR.NORM@**, and the value of POS.ERR@ is set to zero.

POS.MOD@ **Position Modulus** ORMEC Variable

Purpose: Define a **modulo position** conversion factor so that axis position variables **POS.CMD@** and **POS.ACT@** will automatically "repeat" over a specified distance cycle.

Syntax: POS.MOD@[(***axes***)] = *modulus*

Usage: ***var&*** = POS.MOD@[(***axes***)]

modulus A positive **long integer** expression, in the range of **minimum modulus** to **maximum modulus**, which represents the number of **axis position units** over which **POS.CMD@** and **POS.ACT@** will repeat. With the default value of 0, **modulo position** conversion is disabled. When POS.MOD@<>0, only positive values are valid for POS.CMD@ & POS.ACT@, in the range from 0 to POS.MOD@-1.

The minimum assignable value, **minimum modulus**, is the same as the "maximum change in position" per tick, and is given by the following:

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

$$\textbf{minimum modulus} = \frac{\max \Delta pos}{tick} = \frac{\textbf{MTR.SPD.LIM@} * \textbf{CNT.REV@} * \textbf{POS.DIV@}}{60 * \textbf{LOOP.RATE@} * \textbf{POS.MUL@}}$$

This calculation may result in a decimal. **Minimum modulus** is the smallest integer greater than or equal to that number.

The maximum assignable value, **maximum modulus**, is the same as the axis maximum position in **user units**. This is given by the following:

maximum modulus = max pos = **INT** (2147483647 * **POS.DIV@** / **POS.MUL@**)

Modulo position affects the operation of the **MOVE** command. The details of these effects is best described through the following examples:

Examples:

If POS.ACT@=0, POS.MOD@=360, and the axis is commanded to **MOVE** FOR 450, it will move for a distance of 450, but the resulting position will be 90. If it was again commanded to MOVE FOR 450, the resulting position would be 180.

If it was at 0 and received a command to MOVE FOR -450, then the resulting position would be 270.

For **MOVE** TO statements, the sign of the specified position is used to determine the direction of motion, since there are no negative positions and the direction of motion must be defined. Special algorithms like "moving in the direction that results in the shortest move" or "moving in a direction to prevent windup" should be programmed in MotionBASIC.

Note that for positive POS.CMD@ (or POS.ACT@) values and POS.MOD@<>0 the returned value is equivalent to "POS.CMD@ (or POS.ACT@) MOD POS.MOD@"... for negative values, it's the same as POS.MOD@+ POS.CMD@ MOD POS.MOD@.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

POS.MUL@/POS.DIV@ **Axis Position Units Multiplier** ORMEC Variable

Purpose: Define the ratio of axis encoder counts to the desired axis position units allowing MotionBASIC statements to refer to axis positions and distances in **user units**.

Syntax: POS.MUL@[(***axes***)] = *mult*
POS.DIV@[(***axes***)] = *div*
Usage: ***var1%*** = POS.MUL@[(***axes***)]
var2% = POS.DIV@[(***axes***)]

mult A numeric expression in the range 1 to 32768, which defines the numerator of the ratio of **axis encoder** counts to **axis position units**. Default POS.MUL@ = 6000.

div A numeric expression in the range 1 to 32768, which defines the denominator of the ratio of axis encoder counts to axis position units. Default POS.DIV@ = 360.

If the axis encoder has 6000 cts/rev, then the default ratio of 6000/360 defines axis position units of degrees.

POS.ZREF@ **Position at Encoder Reference** ORMEC Variable

Purpose: Return a value (in **user units**) equal to the real-time position of the axis encoder when the **ZREF@** variable was last asserted.

Usage: ***var&*** = POS.ZREF@[(***axes***)]

var A **long integer** read-only variable which indicates the **actual position** of the **axis encoder** when the encoder reference input last asserted the **ZREF@** variable.

The operation of this feature requires a latched sensor configuration. **ZREF@** must be configured as a **R**ising or **F**alling Latched Sensor with the **SENS.MODE@** variable.

The value of POS.ZREF@ is latched within one microsecond of receipt of the sensor. If however **ZREF@** is asserted within 20 microseconds of **ASEN@**, **BSEN@** or **DELAY@**, the value of POS.ZREF@ will be equal to the value of **POS.ASEN@**, **POS.BSEN@** or **POS.DELAY@** respectively.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

PPS.MUL@/PPS.DIV@ **Pacer Position Units Multiplier** ORMEC Variable

Purpose: Define the ratio of **pacer** encoder counts to the desired pacer position units allowing MotionBASIC statements for **follower** axes to refer to pacer distances in **user units**.

Syntax: PPS.MUL@[(***axes***)] = *mult*
PPS.DIV@[(***axes***)] = *div*
Usage: ***var1%*** = PPS.MUL@[(***axes***)]
var2% = PPS.DIV@[(***axes***)]

mult An **integer** expression which defines the numerator of the ratio of **pacer** encoder counts to **pacer position units**. Default PPS.MUL@ = 6000.

div An **integer** expression which defines the denominator of tha ratio of pacer encoder counts to pacer position units. Default PPS.DIV@ = 360.

If the pacer encoder has 6000 cts/rev, then the default ratio of 6000/360 defines pacer position units of degrees.

PRINT **Display on Screen** Statement

Purpose: Display one or more numeric or string expressions on screen device 0. (**?** is a synonym for **PRINT**.) See also **PRINT #**.

Typical: [**REPEAT**] PRINT [@*row*[,*col*[,*cursor*]];] [**USING** *format*;][*exprlist*][;]

row A numeric expression in the range 1 to screen max (less rows used for **function key text**). Sets row position of the cursor.

col A numeric expression in the range 1 to **WIDTH**. Sets the column position of cursor. Default is current location.

cursor A numeric expression. 0 for invisible, 1 for visible cursor.

format A string variable or constant specifying the *format* in which the data is to be printed. Refer to **USING** for details.

exprlist Numeric and/or string expressions to print. Each expression is separated from the next by a **space**, **comma**, or a **semicolon**.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

`;` If a semi-colon is included at the end of the statement, it suppresses the usual carriage return and line feed.

PRINT with no exprlist and no terminating semi-colon sends only the usual carriage-return, line-feed pair.

Screen Positioning Rules:
MotionBASIC divides the screen into zones of 14 character positions. If an expression is followed by a **comma**, MotionBASIC prints the next expression at the beginning of the next zone. If an expression is followed by a **semi-colon** or **space character**, MotionBASIC prints the next expression directly after the previous one.

If the last expression in the list to be displayed is followed by a comma, a semicolon, or either the **SPC**, or **TAB** function, MotionBASIC spaces appropriately and suppresses the carriage-return, line-feed pair. Otherwise, MotionBASIC sends a carriage-return and line-feed after the last expression in the PRINT statement.

MotionBASIC assumes a maximum line length of 80 characters, unless a different value has been specified by means of a **WIDTH** statement or a specific driver has been applied with an **OPEN** statement. MotionBASIC sends a carriage-return, line-feed pair when the maximum line length has been reached.

The formal syntax notation for the **PRINT @** statement is shown below:

Syntax:
[**REPEAT**] PRINT @{*row*|[*row*],{*col*|[*col*],cursor}}; [**USING** *format*;][*exprlist*][;]

This allows for some of the more uncommon cursor positioning techniques that are legal in MotionBASIC, such as:

```
PRINT @1,,0;  'at row 1, but current col, print with an invisible cursor
PRINT @,10;   'at current row, but column 10 print with a visible cursor
PRINT @,,0;   'at current cursor location print with an invisible cursor
```


Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

PRINT # **Output to a Device or File** Statement

Purpose: Write to a specified device or **data file** an image of what would be displayed on screen by a corresponding **PRINT** statement.

Syntax: PRINT #*devnum*, [**USING** *format*;] [*exprlist*][;]

devnum The number under which the device was opened for output.

format A string variable or constant specifying the *format* in which the data is to be printed. See **USING** for details.

exprlist Numeric and/or string expressions to print. Each expression must be separated from the one after it by either a comma or a semicolon.

; If included at the end of the statement, suppresses the usual carriage return and line feed.

? is a synonym for the keyword PRINT.

PRINT # outputs exactly the same characters that would appear on screen in response to a PRINT statement. Commas between expressions, for example, result in the insertion of space.

If an expression to be output contains significant quote marks or commas, you should surround it with a quote mark by means of CHR$(34); otherwise, when you input these fields, MotionBASIC will interpret the significant commas or quotes as delimiters.

See Also: **ITM Driver Support**

PUT **Write to a Random Access File** Statement

Purpose: Writes a specified record from a specified random access file buffer to a **random access file** on a **Memory Disk**.

Syntax: PUT [#]*devnum* [,*recnumber*]

devnum An **integer** expression specifying a device from 1 to 15.

recnumber A positive, non-zero long expression specifying the record number to be transferred. Defaults to the next record, or record 1 (if no record has been **PUT**/**GET**).

A **FIELD** must be declared before execution of a PUT statement.

See Also: **GET**, **FIELD**, **LSET**, **RSET**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Purpose: Program memory is divided into **User** and **Setup** program areas; Allow changes to no areas, one area, or both areas.

Syntax: PW@ = *mode*

Usage: ***var%*** = PW@

mode An integer expression which enables part or all of the program memory to be modified. Changes are allowed to program memory lines as follows:

```
0 - NO lines may be changed     POWERUP DEFAULT SETTING
1 - lines from     0 to 39999 (User  program area)
2 - lines from 40000 to 65529 (Setup program area)
3 - lines from     0 to 65529 (Both  program areas)
```

The effect of *PW@* on the operation of **AUTO**, **DELETE**, **LIST**, & **RENUM** is described below:

```
Syntax: AUTO    [startline [,increment]]
        DELETE  [start] [-[end]]
        LIST    [start] [-[end]]
        RENUM   [newstart] [,[[oldstart][-oldend]] [,increment]]
```

This chart shows the defaults used for the above commands.

Defaults change with the current setting of PW@.

▒ shows areas protected from changes

AREA OPEN	None	User	Setup	Both	
start	0	0	▒▒▒	0	
	1	1	▒▒▒	1	oldstart
startline	10	10	▒▒▒	10	newstart
end	39999	39999	▒▒▒		oldend
start	▒▒▒	▒▒▒	40000		oldstart
startline	▒▒▒	▒▒▒	40010		newstart
end	▒▒▒	▒▒▒	65529	65529	oldend

The *extent* parameter defines the range of influence of **EDIT**, **LOAD**, **MERGE** and **NEW** as well as the above commands with the exception of LIST.

AREA OPEN	*PW@*	*Extent*
None	0	none
User	1	0 - 39999
Setup	2	40000 - 65529
Both	3	0 - 65529

RUN and **SAVE** commands are **not** affected by the value of PW@.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

RANDOMIZE **Reseed Random Number Generator** Statement

Purpose: Supply a new seed for the pseudo-random number generator.

Syntax: RANDOMIZE ***intexpr***

expression An **integer** expression which is a seed for the random number generator.

Each random number seed produces a unique sequence of random numbers, but the same seed always produces the same sequence.

See also: **RND**

READ **Assign DATA Items to Variables** Statement

Purpose: Assign values in **DATA** items to specified variables.

Syntax: READ *variable* [,*variable*] ...

variable A numeric or string variable, or an array element.

A single READ statement can assign multiple **DATA** items to multiple variables; or several READ statements can assign the same DATA item to several variables.

The **RESTORE** statement allows the rereading of DATA items.

Unless a RESTORE statement is executed, MotionBASIC moves to the next DATA item with each READ assignment. If MotionBASIC runs out of DATA items to READ, an Out of Data error occurs. If the *variable* and the data item do not match in type, a Syntax Error occurs.

See also: **DATA** **RESTORE** **Program Initialization**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

REM **Comment** Statement

Purpose: Add explanatory commentary to a program listing.

Syntax: REM [*remark*]

remark Commentary text, ignored by MotionBASIC.

MotionBASIC does not execute REM statements, but a program may branch to a REM statement; execution continues with the first executable statement following the REM.

A REM statement must be the last statement in a logical program line MotionBASIC ignores everything after the REM keyword.

A **single-quote** mark (') is **the recommended** equivalent to REM. A single quote can also be used to append a comment to the end of any program line. A REM comment cannot be written this way; it must be in a program statement by itself.

RENUM **Renumber Program Lines** Command

Purpose: Renumber all or part of a program, starting at a specified line number and incrementing each line by a specified amount.

Syntax: RENUM [*newstart*] [,[*oldstart*] [-[*oldend*]]] [,*increment*]

newstart First line number to be used in the new number sequence.
oldstart First line number or **label** of the old line numbers to be renumbered.
oldend Last line number or **label** of the old line numbers to be renumbered. If *oldend* does not exist then MotionBASIC will generate an error.
increment Interval between line numbers in the renumbered program. Defaults to 10.

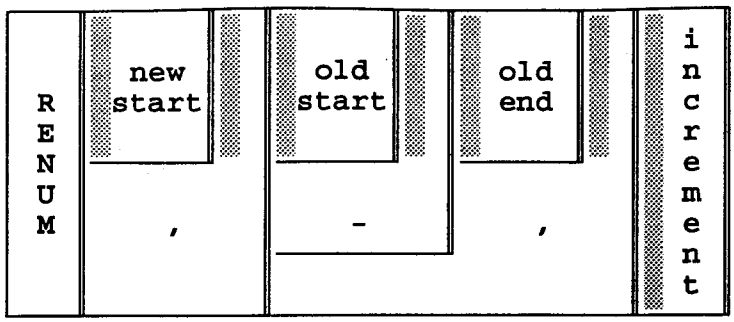
See **PW@** for the default *newstart*, *oldstart* and *oldend* values. RENUM adjusts all line number references in statements using **ELSE**, **GOSUB**, **GOTO**, **ON...GOTO**, **ON...GOSUB**, **RESTORE**, **RESUME**, **THEN** & **ERL**.

RENUM WILL NOT RE-ORDER LINES IN THE PROGRAM; you must use the **text editor** for this purpose.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

RENUM Statement **SYNTAX CHART**

For Instructions See:
Syntax Chart Rules



newstart First line # to be used in the new number sequence.
oldstart First line # or **label** of the old lines to be renumbered.
oldend Last line # or **label** of the old lines to be renumbered.
increment Interval between lines in renumbered program. Default 10.

RENUM CANNOT REPOSITION LINES IN A PROGRAM; For this, use a **Text Editor**

Default Parameters used for RENUM Command

AREA OPEN **PW@=**	User 1	Setup 2	Both 3
oldstart	1		1
newstart	10		10
oldend	39999		
oldstart		40000	
newstart		40010	
oldend		65529	65529

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Purpose: Cause motions specified by **MOVE** or **GEAR** statements to be placed in the DSP **motion queue** where they remain for repetitive execution; cause BASIC to repeat a **PRINT** statement in **direct** mode.

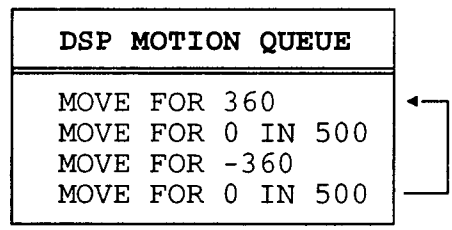
Usage 1: REPEAT **MOVE FOR** *parameters* [AFTER *condition*] [UNTIL *condition*]
Usage 2: REPEAT **GEAR FOR** *parameters* [AFTER *condition*] [UNTIL *condition*]
Usage 3: REPEAT **GEAR AT** *parameters* [AFTER *condition*] UNTIL *condition*
Usage 4: REPEAT **PRINT** *parameters*

condition **ZREF@**, **ASEN@**, **BSEN@** or **DELAY@**

When MotionBASIC executes a **MOVE** or **GEAR** statement, it processes the parameters and places high level information in the **shared memory** of the **DSP Axis Module**(s). When the *REPEAT* modifier precedes the statement, the information is placed in the DSP **motion queue**. The DSP continues to cycle through these motions locally, freeing up MotionBASIC execution for other tasks. REPEAT PRINT is a direct command only, and MotionBASIC loops on that statement until a character is received at the serial port.

Note that since **user units** conversion for **MOVE FOR** and **GEAR FOR** statements is done by the main processor, REPEAT motions for these statements will each cause the commanded position to change by the nearest integral number of encoder counts.

Example:
```
REPEAT MOVE FOR 360
REPEAT MOVE FOR 0 IN 500
REPEAT MOVE FOR -360
REPEAT MOVE FOR 0 IN 500
```



At the DSP level a 360 forward move, 1/2 sec delay, 360 reverse move, and a 1/2 sec delay will continue to execute. A REPEAT queue can be immediately cleared by a **HALT**, **MOVE AT**, or **GEAR AT** statement. It can also be cleared at the end of the sequence by executing a **MOVE FOR 0 IN 0** or a **GEAR FOR 0 in 0**.

See Also: **DSP.CTR@**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

RESTORE **Reset DATA Item Pointer** Statement

Purpose: Reset the **DATA** item pointer to the first item in the first DATA statement or to the first DATA item in a specified DATA statement.

Syntax: RESTORE [*linenum* | **label**]

linenum A program line number.

label A symbolic **label** at the beginning of an existing program line.

If the statement at *linenum* or *label* is not a DATA statement, BASIC RESTOREs to the first DATA statement following *linenum* or *label*.

See also: **DATA** **READ** **Program Initialization**

RESUME **Continue after Error Trap** Statement

Purpose: Restore error trapping and direct program execution after an error recovery procedure has been invoked.

Syntax 1:	RESUME	Goes back to the statement of error
Syntax 2:	RESUME **NEXT**	Goes to the statement after error
Syntax 3:	RESUME {*linenum* | **label**}	Goes to a specific program statement

RESUME to a specific label, Syntax 3, is the recommended approach! RESUME, without an argument, causes the program to try again at the statement that was executing when the error occurred.

NEXT RESUME **NEXT** causes program execution to continue at the statement following the statement where the error occurred. This is **NOT RECOMMENDED**, due to the following: An **IF** block conditional test could be executing when an error occurs. A RESUME NEXT from the error handler would restore execution to the next physical line, regardless to the normal program flow that the IF block's conditional test might have directed.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

linenum RESUME *linenum* or RESUME *label* causes program execution to
label continue at a specified line number or program label.

All ON ERROR **GOTO routines should contain a RESUME statement.** After an Error Handler routine has been invoked, error trapping is suspended until a RESUME is executed. Before a RESUME is executed the program is vulnerable to a second error that would end program execution and return MotionBASIC to **direct** mode.

If program flow is not directed to RESUME operation at the lowest level **FOR...NEXT**, **WHILE...WEND** or **subroutine** where the error occurred, then the MotionBASIC control stack should be cleared. After a **STACK CLEAR** statement, program operation can then be restarted at any location with a **GOTO** statement. If a **STACK CLEAR** is not done, then it is likely that the MotionBASIC control stack will grow in size, continually reducing the amount of memory available. Eventually Error **#1801 Out of Memory** occurs.

A RESUME executed outside an error handler routine generates Error **#1110**.

An example program flow for an Error Handler:

```
ERR.HDLR:                                  'Previously defined in ON ERROR statement
'  user code                               'Here program is vulnerable to an error
'  user code
RESUME MORE.ERROR.HDLR                     'Error trapping is re-established
                                           'and execution directed to additional
MORE.ERROR.HDLR:                           'code to rectify the current error
'Typically used functions &
'variables in error handlers
'include:
'  ERR    ERL    ERR$
'  FAULT@     AFAULT@
'  ALARM@  AXIS.FLT1@
'  AXIS.FAULT@
STACK CLEAR                                'performed since a restart will be done
GOTO RESTART.MY.MACHINE                    'send execution to a restart routine
```

RETURN **Return from Subroutine** Statement

Purpose: Transfer program execution from a subroutine to the line following the GOSUB, or to a specified line.

Syntax: RETURN [*linenum* | **label**]

linenum A program line number.

label A symbolic **label** at the beginning of a program line.

See also: **GOSUB**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

RIGHT$ **Substring at Right** Function

Purpose: Return a specified number of characters from the right side of a string.

Usage: ***var$*** = RIGHT$(*stringexpr*,*length*)

stringexpr A string expression.

length An **integer expression** in the range 0 to 255 which specifies the number of characters to be returned. If length is larger than **LEN**(*stringexpr*), RIGHT$ returns all of *stringexpr*. If length is 0, RIGHT$ returns a null string.

See also: **LEFT$** **MID$**

RMDIR **Delete a directory on the Memory Disk** Command

Purpose: Deletes a directory that currently exists on the **Memory Disk**. The directory to be deleted **must be empty of all files and sub-directories** before it can be deleted.

Syntax: RMDIR "[1:]**path**"

path A string expression that follows DOS path specification conventions to specify the name of the directory to be deleted.

No error is generated when attempting to remove a directory that is non-existant.

See also: **CHDIR** **MKDIR** **FILES**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

RND **Random Number** Function

Purpose: Generate an pseudo-random number in the range of 0 to 1.

Usage: ***var#*** = RND(***numexpr***)

If *numexpr* is:

positive, RND returns the next random number in the current sequence;

zero, RND returns the last (most recent) random number in the current sequence;

negative, MotionBASIC uses *numexpr* to reseed the random number function. The first number in the reseeded random number series is returned.

See also: **RANDOMIZE**

RSET **Right-Justify String Data** Statement

Purpose: Right-justifies string data in a specified field of the random buffer, in preparation for a **PUT** statement or right-justifies string data in ordinary **string variables**.

Syntax: RSET *strvar* = *stringexpr*

strvar A valid string variable name; specifies the variable into which *stringexpr* is to be right justified.

stringexpr A string expression.

If **LEN**(stringexpr) is less than **LEN**(fieldvar), fieldvar is padded on the left with space characters.

See Also: **LSET**, **PUT**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

RUN **Execute Program** Command

Purpose: Execute the current program from the beginning or from a specified line number or label.

Syntax: RUN [*linenum* | **label**]

linenum A program line number.

label A symbolic **label** at the beginning of a program line.

If no line argument is given, MotionBASIC begins execution at the lowest numbered line.

Note: To preserve the contents of non-volatile variables, MotionBASIC does not clear all program variables when a RUN command is executed. The **CLEAR** statement may be used to clear all variables.

See Also: **GOTO** **Program Initialization** **Differences from GWBASIC**

SAVE **Write Current Program to Disk** Command

Purpose: Save the controller's entire program buffer into a specified file

Syntax 1: SAVE *pcfile* [,{ C | P }] On Development System **Disk**
Syntax 2: SAVE *cardfile* [,{ C | P }] On **Memory Disk**

pcfile "[*drive*:][*path*\]*filename*[.*ext*]"
cardfile "1:[*path*\]*filename*[.*ext*]"

drive Development Computer **disk** drive letter.
path A string expression that follows **DOS file-naming**
filename conventions.
ext Default extension is **.BAS** (or **.BPS** when encrypted)

Two **program encryption** options are available when saving program files.

WARNING!!! DO NOT SAVE ENCRYPTION ONTO YOUR ORIGINAL SOURCE FILE NAME!! YOU WILL LOSE YOUR WORK AND NOT BE ABLE TO RECOVER IT!!

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

The **program encryption** options available when saving program files are:

C Creates a CODED file that is unreadable with text editors. When loaded into a controller, it can be viewed and traced using the **LIST** and **TRACE** commands, however no changes can be made to the program. The controller's program buffer is now WRITE PROTECTED.

P Creates a CODED file that is unreadable with text editors. When loaded into a controller, it cannot be viewed at all, and no changes can be made to the program. The controller's program buffer is now WRITE and READ PROTECTED.

WARNING!!! DO NOT SAVE ENCRYPTION ONTO YOUR ORIGINAL SOURCE FILE NAME!! YOU WILL LOSE YOUR WORK AND NOT BE ABLE TO RECOVER IT!!

```
Example: SAVE "C:\ORMEC\BAS\TEST",C      'Puts TEST.BPS on the Hard Drive
```

See Also: **LOAD** **MERGE** **AUTOLOAD.BAS** **PB1LOAD.BAS**

SCURVE@ **S-Curve Velocity Profile** ORMEC Variable

Purpose: Provide the ability to adjust the "smoothness" of velocity ramps commanded by **MOVE** statements. As SCURVE@ increases, the peak value of "jerk",(instantaneous acceleration change), is reduced.

Syntax: SCURVE@[(***axes***)] = *smoothness*
Usage: ***var%*** = SCURVE@[(***axes***)]

smoothness An **integer expression** in the range 0 to 100; specifies the percentage of a velocity ramp to be "smoothed"; Default= 0

In linear velocity ramps, SCURVE@=0, the required acceleration is applied, held constant over 100% of the ramp, and removed. **NOTE:** Ramps taking 5 seconds or more are commanded as linear, regardless of SCURVE@.

When SCURVE@ is used, the velocity ramp is portioned into three segments: a smoothed start, a linear mid-section, and a smoothed finish. The length of the segments are percentages of the total velocity ramp, respectively they are: (SCURVE@/2)%, (100-SCURVE@)%, (SCURVE@/2)%.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

For example, **SCURVE@(1) = 50**, and a **MOVE 1 AT 3000 IN 4000** is commanded. This defines a velocity ramp to a speed of 3000, over 4 seconds. With SCURVE@=50, the smoothed start is 25% (1 sec), the linear mid-section is 50% (2 secs), and the smoothed end of the ramp is 25% (1 sec).

An S-Curve velocity ramp is produced as follows:
Acceleration is increased from zero to the necessary peak value during the first segment, held constant over the second segment, and then decreased back to zero acceleration during the third segment. This produces a "smoothed" velocity profile with a linear mid-section.

When using the S-Curve feature, **the peak acceleration**, (used during the linear mid-section), **is higher** than the acceleration that would have been used without S-Curve. In general terms, a short period of "higher" acceleration is needed to compensate for the lower values used in the beginning, and end, of the smoothed velocity ramp.

As SCURVE@ increases, so does the peak acceleration requirement.

SENS.MODE@ **Sensor Mode** ORMEC Variable

Purpose: Select the mode of operation of the high speed sensors provided on the **DSP Axis Module.**

Syntax: SENS.MODE@[(***axes***)] = "*mode*[*agate*[*bgate*]]" Set Configuration
Usage: ***var$*** = SENS.MODE@[(***axes***)] Get Configuration

mode A three character string expression combining the characters "**H**", "**L**", "**R**", and "**F**", which sequentially define the **ZREF@**, **ASEN@** & **BSEN@** inputs as either asserted **H**igh, **L**ow, on a **R**ising edge, or on a **F**alling edge. Default is "RRR".

agate One character each, either an "**N**" or "**G**" which will
bgate disable/enable **ELS** gating of **ASEN@** and **BSEN@** respectively. **N**ot Gating, or **G**ating. Default is "NN".

var$ When read a 3, 4, or 5 character string is returned indicating the respective operating modes of **ZREF@**, **ASEN@** & **BSEN@**, and any gating that has been enabled.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

SENS.MODE@ **Sensor Mode** ORMEC Variable

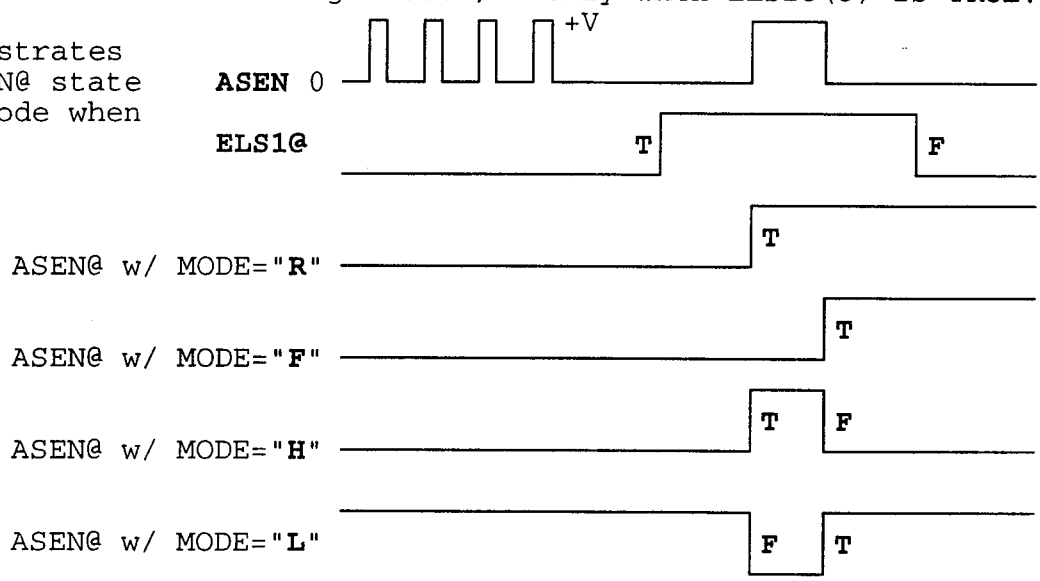
Sensor	For Which Axis?	Gated by Which ELS?
ASEN@	first on DSP Axis Module	ELS1@(first)
BSEN@	first on DSP Axis Module	ELS2@(second)
ASEN@	second on DSP Axis Module	ELS1@(second)
BSEN@	second on DSP Axis Module	ELS2@(first)

Example: For a Dual DSP module controlling Axis 3 & Axis 4

```
10 CONFIGURE.SENSORS:
20 SENS.MODE@({4}) = "RFHGG"
30 PRINT SENS.MODE@({4})
40 RETURN
GOSUB CONFIGURE.SENSORS
RFHGG
```

The example above sets up Axis 4 to have the following sensor action:
ZREF@ to be asserted on a rising edge.
ASEN@ to be asserted on a falling edge, only when **ELS1@**(4) is **TRUE**.
BSEN@ to be asserted on a high level, only when **ELS2@**(3) is **TRUE**.

This diagram illustrates the resultant ASEN@ state for each sensor mode when gating is used.



SGN **Sign of Number** Function

Purpose: Return the sign of its numeric argument.

Usage: ***var%*** = SGN(***numexpr***)

If *numexpr* is: positive, SGN(*numexpr*) returns 1;
zero, SGN(*numexpr*) returns 0;
negative, SGN(*numexpr*) returns -1.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

SIN **Sine** Function

Purpose: Return the sine of the angle whose value, in radians, is expressed in *numexpr*.

Usage: ***var#*** = SIN(***numexpr***)

numexpr A numeric expression which represents the angle, in radians. If *numexpr* is large (greater than 1.0E8), a partial loss of significance in the result may occur. If *numexpr* is so large that a total loss of significance occurs (greater than 2.0E9), an Overflow error will occur.

Note: PI radians = 180 degrees, and so before executing the sine function, convert numexpr from degrees to radians by dividing by 180/PI or 57.296.

See also: **ATN** **COS** **TAN**

SPACE$ **String of n Spaces** Function

Purpose: Return a string consisting of a specified number of space characters.

Usage: ***var$*** = SPACE$(*n*)

n An **integer expression** in the range 0 to 255 specifying the number of spaces to return to *var$*.

See also: **SPC** **STRING$** **TAB**

SPC **Output n Spaces in a PRINT Statement** Modifier

Purpose: Output a specified number of spaces in a **PRINT** or **PRINT #** statement.

Usage: PRINT SPC(*n*)

n An **integer expression** in the range 0 to 255.

If *n* is larger than the width of the output device (as defined by WIDTH), MotionBASIC skips *n* **MOD** width spaces.

If the SPC function is used at the end of a list of items to print, MotionBASIC behaves as though the PRINT statement terminated with a semicolon; it suppresses the usual carriage return and line feed.

See also: **SPACE$** **TAB** **WIDTH**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

SPD.MAX@ **Maximum Speed** ORMEC Variable

Purpose: Define the maximum allowable commanded speed in **user units.** This is also the default speed used by motion statements.

Syntax: SPD.MAX@[(***axes***)] = *speed*

Usage: ***var&*** = SPD.MAX@[(***axes***)]

speed A **long integer** expression representing the maximum allowable commanded axis speed in user units from 1 to **USR.SPD.LIM@.** Default SPD.MAX@ = USR.SPD.LIM@, and **changing** USR.SPD.LIM@ **will always reset SPD.MAX@ equal to USR.SPD.LIM@.**

This speed is used by default for all motion statements. If a motion statement specifies a speed greater than this parameter, an error will be generated and the motion will not occur. This variable can be changed at any time without affecting any other variables, but must always be less than or equal to **USR.SPD.LIM@.**

SQR **Square Root** Function

Purpose: Return the square root of a number.

Usage: ***var#*** = SQR(***numexpr***)

numexpr A numeric expression greater than or equal to 0.

STACK **Clear Control Stack** Statement

Purpose: Clear the MotionBASIC control stack, aborting all loops and subroutines, and allowing the program to start over properly.

Usage: STACK CLEAR

The MotionBASIC control stack is used to keep track of nested **subroutines**, **FOR...NEXT** and **WHILE...WEND** loops.

The STACK CLEAR statement clears the status of the MotionBASIC control stack causing all program loop and subroutine returns to be aborted. After a STACK CLEAR statement, program operation is normally continued at the main program entry point with a **GOTO** statement.

See also: **RESUME**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

STL.FWD@ **Forward Software Travel Limit** ORMEC Variable

Purpose: Define the maximum allowed forward position for a servo.

Syntax: STL.FWD@[(***axes***)] = *position*

Usage: ***var&*** = STL.FWD@[(***axes***)]

position A long numeric expression which represents the maximum allowed forward (positive) position for a specified servo in user units. Setting both STL.FWD@ and **STL.REV@** to 0 (default) disables the use of software travel limits.

If a servo's **actual position** becomes more positive than the "forward software travel limit", then:
- it's commanded motion will be stopped using **USR.ACL.LIM@**,
- the **OTL.FWD@** Boolean variable for that axis will be asserted, and
- a **motion error** will be generated.

See **OTL.FWD@** for a detailed explanation of overtravel limit operation.

STL.REV@ **Reverse Software Travel Limit** ORMEC Variable

Purpose: Define the maximum allowed reverse position for a servo.

Syntax: STL.REV@[(***axes***)] = *position*

Usage: ***var*** = STL.REV@[(***axes***)]

position A long numeric expression which represents the maximum allowed reverse (negative) position for a specified servo in user units. Setting both **STL.FWD@** and STL.REV@ to 0 (default) disables the use of software travel limits.

If a servo's **actual position** becomes more negative than the "reverse software travel limit", then:
- it's commanded motion will be stopped using **USR.ACL.LIM@**,
- the **OTL.REV@** Boolean variable for that axis will be asserted, and
- a **motion error** will be generated.

See **OTL.REV@** for a detailed explanation of overtravel limit operation.

STOP **Halt Program, Leave Devices Open** Statement

Purpose: Stop program execution but leave all devices open.

Syntax: STOP

MotionBASIC reports the line number at which program execution was STOPped.

STOP is generally used for debugging purposes. Program execution can be resumed after a STOP statement by means of a **CONT** command, provided the program has not been edited during the interruption.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

STR$ **String Representation of Numeric Expression** Function

Purpose: Return either the "decimal representation of a number" or the "elements in a set" as a string.

Usage: ***var$*** = STR$(***numexpr***)

The string returned by STR$ will contain no leading or trailing spaces.

To support **sets**, STR$({elem1,elem2,elem3,...}) returns the string "{elem1,elem2,elem3,...}"

See also: **CHR$** **VAL** **HEX$** **OCT$** **BIN$** **Differences** from GWBASIC

STRING$ **String of Specified Length and Character** Function

Purpose: Return a string of specified length, whose characters all have the same specified **ASCII** value.

Usage 1: ***var$*** = STRING$(*m*,*n*)

Usage 2: ***var$*** = STRING$(*m*,*stringexpr*)

m An **integer expression** in the range 0 to 255; specifies the length of the string.

n An **integer expression** in the range 0 to 255; specifies the decimal **ASCII** value of the character to be returned *m* times.

stringexpr A string expression. STRING$(*m*,*stringexpr*) returns the first character of *stringexpr* *m* times.

SYSTEM **Return to Operating System (BASIC)** Command

Purpose: **None. This keyword is currently reserved for future use.**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

TAB **Tab to Specified Position** Modifier

Purpose: Move to a specified print position in a **PRINT** or **PRINT #** statement.

Usage: PRINT TAB(*column*)

column An **integer expression** in the range 1 to 255.

If the current print (or screen) position is already to the right of *column*, TAB moves to position *column* on the next line.

If the TAB function comes at the end of a list of data items in a PRINT statement, MotionBASIC behaves as if the PRINT statement were terminated by a semicolon; it suppresses the usual carriage return and line feed.

If *column* is larger than the width of output device, TAB(*column*) moves to *column* MOD width.

If *column* is less than 1, TAB(*column*) is treated as TAB(1).

See also: **SPACE$** **SPC**

TALK **Console Talk Mode** Statement

Purpose: Put the console into "TALK" mode with the devices specified on the command line. This means that all characters typed at the console will be sent to the device specified by the *outdev* number and all characters received from the device specified by the *indev* number will be displayed on the console.

Syntax: TALK *indev, outdev*

indev A numeric expression in the range 1 to the maximum number of devices allowed.

outdev A numeric expression in the range 1 to the maximum number of devices allowed.

The "TALK" mode is terminated by typing a Ctrl-C at the console.

See also: **OPEN** **EOF** **INPUT$**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

TAN **Tangent** Function

Purpose: Returns the tangent of the angle whose value, in radians, is expressed in *numexpr*.

Usage: ***var#*** = TAN(***numexpr***)

numexpr A numeric expression which represents the angle, in radians. If *numexpr* is large (greater than 1.0E8), a partial loss of significance in the result may occur. If *numexpr* is so large that a total loss of significance occurs (greater than 2.0E9), an Overflow error will occur.

PI radians = 180 degrees, and so before executing the tangent function, convert numexpr from degrees to radians by dividing by 180/PI or 57.296.

See also: **ATN** **COS** **SIN**

TIME$ **Set System Time or Return it as a String** Function

Purpose: Set system time or return it as a string.

Syntax: TIME$ = *stringexpr* Set the system time

Usage: ***var$*** = TIME$ Get the system time

stringexpr A string expression in the form "hh" or "hh:mm" or "hh:mm:ss". Leading 0s may be omitted in any of the fields.

var$ The value returned will be an 8-character string in the form "hh:mm:ss".

Example:
```
10 TIME$ = "0"          'reset the system time
```

See also: **TIMER**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

TIME.MUL@ **Time Conversion Factor** ORMEC Variable

Purpose: Define the number of internal time units (1 millisecond each) per desired user time unit.

Syntax: TIME.MUL@ = *units*

Usage: ***var%*** = TIME.MUL@

units An integer expression which defines the number of internal time units (1 millisecond each) per desired user time unit.

The default for TIME.MUL@ is 1 which provides user time units of 1 ms for all references to time in a MotionBASIC program.

Example: To specify time in ten-thousandths of a minute, change this multiplier to 60*1000/10000 = 6.

TIMER **Return elapsed time in User Units** Function

Purpose: Return elapsed time in **user units.**

Usage: ***var&*** = TIMER Get the system time

var& The **long integer** value returned will contain the time in **user units** since the system time was last reset.

Example:

```
10 START.TIME=TIMER                                  'save current time
20 WAIT 100
30 PRINT "Elapsed Time:";TIMER-START.TIME;"ms"       'print the elapsed time
40 END
```

TRACE **Program Trace Control** Statement

Purpose: Enable or disable the program trace mode.

Syntax 1: TRACE [SAVE] {ON | OFF | STEP}
Syntax 2: TRACE RESTORE

ON Causes MotionBASIC to output a copy of the program line before it is executed.

OFF Causes MotionBASIC to stop tracing the program.

STEP Same as ON except that MotionBASIC waits for a keystroke before the execution of the printed statement.

In trace mode, MotionBASIC prints the line, enclosed in square brackets, of each program line as it is executed.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

The option **SAVE** keyword will save one level of the setting of TRACE. In other words if you have a particularly annoying piece of code for tracing you can do a TRACE SAVE OFF before it and a TRACE RESTORE after it right in the program buffer. This will prevent unnecessary tracing of the code.

TRACE may be executed in either direct mode or as part of a program execution.

See also: **BREAK** **TRACE VAL** **Program Initialization**

TRACE VAL — Variable Trace Control — Statement

Purpose: Enable or disable the variable trace mode.

Syntax 1: TRACE VAL [ON] *varlist*
Syntax 2: TRACE VAL OFF [*varlist*]
Syntax 3: TRACE VAL LIST

ON Causes MotionBASIC to trace on the specified variables.

OFF Causes MotionBASIC to stop tracing those specified variables or all variables.

LIST Causes MotionBASIC to list all variables currently being traced.

varlist A list of variables (separated by commas) which, when modified during program execution, will cause MotionBASIC to process the trace.

In variable trace mode, as program lines are executed, MotionBASIC prints the line, enclosed in square brackets, of each program line which changes one of the variables in the list.

TRACE VAL may be executed in either direct mode or as part of a program execution.

When used with ORMEC variables which are indexed by **axes**, the axes can not be specified in the *varlist*. TRACE VAL will display changes to that ORMEC variable on all axes.

See also: **BREAK** **CLEAR** **TRACE** **Program Initialization**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

DRV.CMD@ **Drive Command** ORMEC Variable

Purpose: Return the current real-time value for the drive command output of an **axis control loop**. Set the drive command output directly when in **output mode**. (Formerly known as **TRQ.CMD@** for MB ver 2.0 and prior.)

Usage: ***var%*** = DRV.CMD@[(***axes***)]

Syntax: DRV.CMD@[(***axes***)] = **intexpr**

var An **integer** value within the range from -**DRV.MAX@** to DRV.MAX@, corresponding to the current drive command output. The value is proportional to ±10 VDC, and is positive for "forward" output.

intexpr An **integer expression** which will directly set the drive command output voltage (DRV-CMD) from -10 to +10 VDC when in **output mode**.

The DSP Axis Module loop hardware output (DRV-CMD) is interfaced through a 12-bit D/A converter and only the most significant 12 bits of the DRV.CMD@ variable affect the output voltage. i.e. The analog output voltage is changed only at increments of 16 in the value of DRV.CMD@.

The value for the DRV.CMD@ variable is always positive for "forward" commands. For Generation III drives, motor rotation should be "counter-clockwise" for a "positive" voltage input to the servodrive. When the axis direction for "forward" is changed to "clockwise", by assigning a non-zero value to **CW.FWD@**, the polarity of the drive command output voltage is reversed with respect to the DRV.CMD@ variable. Also reversed at that time is the direction of the servomotor feedback.

The Drive Command output is *normally changed indirectly* by MotionBASIC **motion statements**, and this variable is only for checking status. However, in **output mode** this variable *can only be* set directly resulting in a "step" change of drive command voltage to the servodrive.

See also: **DRV.MAX@**

TRQ.GAIN@ **Torque Gain of Motor & Drive** ORMEC Variable

Purpose: Define the torque gain of the motor and drive for a servo axis.

Syntax: TRQ.GAIN@[(***axes***)] = *gain*

Usage: ***var!*** = TRQ.GAIN@[(***axes***)]

gain A **single precision** floating point expression which represents the torque gain of the motor and drive from 0.10 to 655.35 lb-in per volt. Default = 0.5

Note that no changes will be made to the actual servo loop parameters after changing TRQ.GAIN@ until a value is assigned to INERTIA@.

See Also: **Loop Architecture** **Ka** **Kaf**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

DRV.MAX@ **Maximum Drive Command** ORMEC Variable

Purpose: Define the maximum allowable value for the **drive command** output. (Formerly known as **TRQ.MAX@** for MB ver 2.0 and prior)

Syntax: DRV.MAX@[(***axes***)] = *maximum*
Usage: ***var%*** = DRV.MAX@[(***axes***)]

maximum An **integer expression** from 1 to 32,767, that defines the maximum **drive command** output value in either direction and proportionally limits the DSP's 12-bit D/A converter output voltage in a range of 0 to ±10 volts. The default value is 1 as a safeguard against any attempt to operate a servo axis without first executing the **MP.CONFIG** routine.

With ORMEC's E-Series and S-Series "torque mode servodrives", a setting of 10000 corresponds to approximately 100.00% of "rated output torque".

With ORMEC's F-Series "torque mode servodrives", a setting of 26214 corresponds to approximately 100% of "peak output torque".

The fact that the drive command output is being limited by this variable will not by itself cause any error condition.

UCASE$ **Convert Lower Cases to Upper Cases** Function

Purpose: Return a string with all lower case characters of its argument being converted into upper case characters.

Usage: ***var$*** = UCASE$(*stringexpr*)

stringexpr A string expression.

See also: **LCASE$**

UNIT.ID@ **Unit Identification Code** ORMEC Variable

Purpose: Return the motion controller's unit identification code.

Usage: ***var%*** = UNIT.ID@

var An **integer** value which is equal to the value selected by the Unit-ID switches on the front of the controller when power was applied to the unit, or the reset button was last pressed.

If **UNIT.ID@** is outside the range of 1-32 at powerup, an **Invalid Unit ID** fault will be generated and a bit in the **FAULT@** variable will be set.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Purpose: Define a stop condition for a motion specified by a **MOVE**, **GEAR** or **HALT** statement; define a terminate condition for a **WAIT** statement.

Usage 1: [REPEAT] **MOVE** *parameters* [AFTER *condition*] UNTIL *condition*

Usage 2: [REPEAT] **GEAR** *parameters* [AFTER *condition*] UNTIL *condition*

Usage 3: **HALT** *parameters* [AFTER *condition*] UNTIL *condition*

Usage 4: **WAIT** UNTIL *expression*

condition **ZREF@**, **ASEN@**, **BSEN@** or **DELAY@**

expression Any **Boolean** expression.

When MotionBASIC executes a **MOVE**, **GEAR** or **HALT** statement, it processes the parameters and places high level information in the **shared memory** of the **DSP Axis Module**(s). When **AFTER**, *UNTIL* and/or **REPEAT** modifiers are included in the statement, the DSP uses this information as it executes or "repeats" the motion, allowing "fast" decision making by the next **servo loop update**. Note that motions modified by **AFTER** and **UNTIL** clauses can be "repeated" independently by the DSP, freeing up MotionBASIC execution for other tasks.

MOVE, **GEAR** and **HALT** statements implicitly clear latched variables when executed, causing the UNTIL clause to be satisfied on the next event.

Note that *UNTIL* clauses can be used with **GEAR AT** statements but not with **GEAR FOR** statements.

When an *UNTIL* clause is used with a **WAIT** statement, then MotionBASIC repeatedly executes the WAIT statement suspending program execution, except for **ON EVENT** processing, until the WAIT condition becomes **TRUE**.

Examples:
```
10 MOVE AT 500 UNTIL ASEN@
20 HALT IN 30 UNTIL ZREF@
```


Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

USER1.LED@ **User 1 LED ON** ORMEC Variable

Purpose: Allow MotionBASIC to turn on and off the green **"User 1"** LED on the front of the controller; return the status of the User 1 LED.

Syntax: USER1.LED@ = *value*

Usage: ***var%*** = USER1.LED@

value This **Boolean** expression is used to turn the LED ON or OFF. If the expression evaluates to 0 (OFF) the LED will be turned OFF, otherwise it will be turned ON. Default USER1.LED@ = OFF (0).

var% A **Boolean integer** variable which will be TRUE (-1) if the LED is ON. Otherwise it will be FALSE (0).

Note: **User 1** LED indicates that a **powerup** program load from a **Memory Disk** is in progress by flashing during the load.

USER2.LED@ **User 2 LED ON** ORMEC Variable

Purpose: Allow MotionBASIC to turn on and off the green **"User 2"** LED on the front of the controller; return the status of the User 2 LED.

Syntax: USER2.LED@ = *value*

Usage: ***var%*** = USER2.LED@

value This **Boolean** expression is used to turn the LED ON or OFF. If the expression evaluates to 0 (OFF) the LED will be turned OFF, otherwise it will be turned ON. Default USER2.LED@ = OFF (0).

var% A **Boolean integer** variable which will be TRUE (-1) if the LED is ON. Otherwise it will be FALSE (0).

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Purpose: Define a format for the **PRINT**, **PRINT #**, and **INPUT @** statements.

Usage: USING *format*

format A string variable or constant specifying the format in which the data is to be printed.

The following symbols may be used in *format*:

Symbol	Meaning (numeric)	Symbol	Meaning (string/literal)
#	Digit in a number	!	First character of string
.	Decimal point in a number	\ \	First n chars of string
+	Leading/trailing +/- sign	&	Entire string
-	Trailing minus sign		
^^^^	Exponent of a number	_	Print next format char
,	Thousands place separator	other	Print format character

Multiple field formats may be contained in a single *format* string. Each field's format should be separated from the next by a non-format character. If the expression list contains more expressions than are specified by the format string, the *format* string will be replicated as many times as are necessary to print all expressions in the list.

For example, the following two statements are equivalent:

```
PRINT USING "##.# ##.### "; 1; 2; 3; 4
 1.0  2.000  3.0  4.000
PRINT USING "##.# ##.### ##.# ##.### "; 1; 2; 3; 4
 1.0  2.000  3.0  4.000
```

The following pages contain a detailed description of each of the format symbols. Each page describes one format character, and contains examples of its use.

See also: **PRINT** **PRINT #** **INPUT @**

Format Symbol: **#**

Meaning: A place-holder. If the numeric expression has more digits to the right of the decimal point than the format string has #s, MotionBASIC rounds. If the numeric expression has more digits to the left of the decimal point than the format string has #s, MotionBASIC prints all the digits to the left of the decimal point and also prints a percent sign (%) to the left of the number. If the numeric expression has fewer digits to the left of the decimal point than the format string has #s, MotionBASIC right-justifies the number; that is, it pads the number on the left with spaces. (But if there are any #s to the left of the decimal point in the format string, MotionBASIC always prints at least one digit -- a 0 if necessary -- to the left of the decimal point).

Example:

```
PRINT USING "#####"; -123
 -123
```


Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Format Symbol: . (decimal point)

Meaning: Specifies the position of the decimal point in a numeric expression.

Examples:
```
PRINT USING "###.##"; 12.3
 12.30
PRINT USING "###.##"; -1.234
 -1.23
PRINT USING "####.##"; 12.345
  12.35
PRINT USING "###.#####"; 0
  0.00000
PRINT USING "###.##"; -37.8; 14.678; 2
-37.80 14.68  2.00
PRINT USING "#.##"; 12.3                    'Example of width exceeded
%12.30
```

Format Symbol: +

Meaning: Print a plus or minus sign, as appropriate, to the left or right of the number. The sign is always printed immediately to the left or right of the number, depending on the position of plus sign.

Examples:
```
PRINT USING "+###.##"; 12.3; -45.6
 +12.30 -45.60
PRINT USING "##.#+"; 2; -7
 2.0+ 7.0-
```

Format Symbol: - (minus sign)

Meaning: Print a minus sign immediately to the right of a negative number (the minus sign should appear after the place holders in the format string). If the number not negative, print a space instead.

Examples:
```
PRINT USING "##.#-"; 2; -7
 2.0  7.0-
```

Format Symbol: ^^^^^ (five carets)

Meaning: Print a number in exponential format. The five carets should appear after all place holders in the format string. The five carets are place holders for E±nnn.

Examples:
```
PRINT USING "##.##^^^^^"; 123.45
 1.23E+002
PRINT USING ".####^^^^^"; 555555
.5556E+006
PRINT USING "+.##^^^^^"; 876
+.88E+003
```


Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Format Symbol: **!**

Meaning: Print only the first character of a string expression.

Example:
```
10 A$="HELLO":B$="WORLD"
20 PRINT USING "!";A$;B$                    'Print "HW"
RUN
HW
```

Format Symbol: **\ **

Meaning: Print only the first *n* + 2 characters of a string expression, where *n* is the number of spaces between the two backslashes. If *n* is larger than the number of characters in the string expression, MotionBASIC pads the string expression on the right with space characters.

Example:
```
10 A$="TRY":B$="THIS"
20 PRINT USING "!";A$;B$                    'Print "TT"
30 PRINT USING "\ \";A$;B$                  'Print "TRYTHI"
40 PRINT USING "\   \";A$;B$;"!!"           'Print "TRY THIS!!"
RUN
TT
TRYTHI
TRY THIS!!
```

Format Symbol: **&**

Meaning: Print a string expression without reformatting it.

Example:
```
10 A$="HELLO":B$="WORLD"
20 PRINT USING "!";A$;                      'Print "H"
30 PRINT USING "&";B$                       'Print "WORLD"
RUN
HWORLD
```

Format Symbol: _ (underscore)

Meaning: Print next character as a literal. The combination _#, for example, allows you to include a number sign as a literal in your numeric format.

Examples:
```
PRINT USING "_!##.##_!"; 12.34
!12.34!
PRINT USING "_######"; 123
#  123
```


Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

USING **Define a format for PRINT or INPUT** Modifier

Format Symbol: (other)

Meaning: Characters other than the foregoing may be included as literals in the format string. Thus, for example, a single dollar sign may be positioned to the left of a series of place holders (#s) to achieve vertically aligned dollar signs, and space characters may be placed at the right side of the format string to achieve horizontal separation between a series of numbers.

Example:
```
PRINT USING "[##.##]"; 12.34
[12.34]
PRINT USING "TOTAL: ####.# (###%)"; 273.4; 78
TOTAL:  273.4 ( 78%)
```

USR.ACL.LIM@ **Acceleration Limit in User Units** ORMEC Variable

Purpose: Define the absolute maximum machine acceleration rate in **user acceleration units.**

Syntax: USR.ACL.LIM@[(***axes***)] = *accel*

Usage: ***var***& = USR.ACL.LIM@[(***axes***)]

accel A **long integer** expression representing the absolute maximum machine acceleration rate in user units. Default USR.ACL.LIM@ = 5000 rad/sec².

This variable is used in conjunction with **MTR.ACL.LIM@** and **CNT.REV@** to define the **user acceleration units.** It is also used whenever software or hardware overtravel limits are exceeded.

This setup parameter should generally be set only once during program initialization.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

USR.SPD.LIM@ **Axis Speed Limit in User Units** ORMEC Variable

Purpose: Define the absolute machine speed limit for a servo axis in **user speed units.**

Syntax: USR.SPD.LIM@[(***axes***)] = *speed*

Usage: ***var***& = USR.SPD.LIM@[(***axes***)]

speed A **long integer** expression representing the absolute maximum machine speed for a servo axis in user units. Default USR.SPD.LIM@ = 4000 RPM.

Note that MotionBASIC sometimes adjusts the value of **MTR.SPD.LIM@** in the process of calibrating the DSP speed equations, and therefore you may need to adjust USR.SPD.LIM@ accordingly. **MotionPRO**'s screen-oriented system configuration capability makes this adjustment for you automatically.

VAL **Numeric Value of a String** Function

Purpose: Return the numeric value represented by a string expression.

Usage: ***var*** = VAL(*stringexpr*)

stringexpr A string expression.

VAL extracts a number from the first characters of its argument. It ignores leading blanks, tabs, and line feeds, and treats a single hyphen as a signifying negation. For example, VAL(" -32 Main Street") returns -32. If the first character in the argument (or the first character following all space and tab characters) is a letter, VAL returns 0. For example, VAL("HI Mom!") returns 0.

To support **sets**, VAL("{elem1,elem2,elem3,...}") returns a set containing elem1,elem2,elem3,...

See also: **ASC** **STR$**

VEL.ACT@ **Actual Velocity** ORMEC Variable

Purpose: Return the observed velocity of the servo axis in user units.

Usage: ***var***& = VEL.ACT@[(***axes***)]

var A **long integer** value which is equal to the current real-time velocity of the servo axis, as determined by the velocity observer.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

VEL.CMD@ **Velocity Command** ORMEC Variable

Purpose: Return the current real-time value for commanded velocity of a servo axis in user units. Set the commanded velocity directly when in velocity mode.

Usage: ***var&*** = VEL.CMD@[(***axes***)]

Syntax: VEL.CMD@[(***axes***)] = **lngexpr**

var A **long integer** value which is the currently commanded velocity.

lngexpr A **long integer** expression which will directly set the commanded velocity if in velocity mode. See **MODE@**.

Commanded velocity is *normally* changed by MotionBASIC statements such as **MOVE@** and **GEAR@**, and this variable is only for checking status. However, in velocity mode this variable *can be* set directly resulting in a "step" change of velocity command fed directly to the velocity loop.

VEL.LAST@ **Last Specified Top Velocity** ORMEC Variable

Purpose: This read-only variable is the top velocity last specified in a **MOVE** command, either directly or indirectly.

Usage: ***var%*** = VEL.LAST@[(***axes***)]

Specifying a top velocity in any MOVE command, either directly or indirectly, will change this parameter.

VLTC@ **Velocity Loop Time Constant** ORMEC Variable

Purpose: Define the velocity loop time constant for a servo axis.

Syntax: VLTC@[(***axes***)] = *factor*

Usage: ***var!*** = VLTC@[(***axes***)]

factor A **single precision** floating point expression which represents the velocity loop time constant from 0.3 to 30.0 msec. Default = 3.0

See Also: **Loop Architecture** **Kv** **Kvi** **Kp** **Kpi**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

WAIT **Wait for a Condition or Time** Statement

Purpose: Suspend MotionBASIC execution to pause until a condition becomes true, for a specified length of time, or for a specified length of time after a condition becomes true.

Syntax 1: WAIT **UNTIL** *condition*
Syntax 2: WAIT *time* [**AFTER** *condition*]

condition An **expression** that must evaluate to **TRUE** before the WAIT time begins or program execution continues.

time A numeric expression representing the time in **user units** that MotionBASIC execution will pause AFTER the specified *condition* evaluates **TRUE**. When the *time* expires, program execution continues at the next statement.

MotionBASIC **event trapping** is enabled during WAIT statements. WAIT will wait at least as long as specified, with a 5 msec minimum and a 5 msec resolution. The table below assumes **TIME.MUL@** = 1.

time	Actual wait time
0	0
1 - 5	5 - 10 msec
6 - 10	10 - 15 msec
11 - 15	15 - 20 msec

WAIT statements should not be used in an **ON EVENT** subroutine. If WAIT statement functionality is needed in an ON EVENT subroutine, refer to Field Service Bulletin #95001.

WHILE, WEND **WHILE...WEND Loop** Statement

Purpose: Define a WHILE/WEND loop.

Syntax:
```
WHILE expression
  .
  . [statements]
  .
WEND
```

expression A numeric or **Boolean** expression.

MotionBASIC starts by evaluating *expression*. If *expression* is non-zero (true), the next statement is executed. If *expression* is zero (false), control passes to the first statement following the next WEND statement.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

When MotionBASIC encounters the WEND statement, it reevaluates the *expression* argument to the most recent WHILE. If the argument is still non-zero (true), the process is repeated; otherwise, execution continues at the next statement.

WHILE/WEND loops may be nested to any level of complexity, but there must be a WEND for each WHILE.

MotionBASIC programs should not include **GOTO** statements which branch out of a **WHILE...WEND** loop. See the **structured programming** chapter for details.

See also: **FOR...NEXT** Differences **from GWBASIC**

WIDTH — Set Output Width — Statement

Purpose: Set the line width in characters for an output device.

Syntax: WIDTH [[#]*devnum*,] *width*

devnum A numeric expression which specifies the device **OPEN**ed for output. If not specified, the *width* value will affect the screen device.

width A numeric expression in the range 0 to 255 with the following exceptions:

The range for a screen device, (e.g. **OIT** or **MMI**) is 14 to 132. Default 80. 0 turns off WIDTH support.

The valid range when **ITM Driver Support** is enabled is 15 to 80. The default is 15.

A carriage return is sent to the device when *width* has been reached.

Specifying a WIDTH of 0 or 255 disables the automatic generation of a carriage return.

Normally, every character sent through the output device will increment an internal counter. When the counter reaches the *width* value, MotionBASIC sends a carriage return to the device. There are three special characters that do not simply increment the counter. They are:

ASCII Code	Character Name	Effect on internal counter
0Dh	Carriage Return	Resets counter to 0
0Ah	Line Feed	No effect
08h	Back Space	decrements counter by 1 (no effect at 0)

An interesting feature of the Color **OIT**, which normally has a width of 80 columns, is obtained by setting the WIDTH to 40. This provides double-wide characters for the entire **OIT** display.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

WITHIN **Set Membership** Operator

Purpose: Perform a set membership comparison.

Usage: *set1* WITHIN *set2*

set1, *set2* Set expressions. If *set2* has all the elements in *set1*, the result is -1 (TRUE); otherwise the result is 0 (FALSE).

WITHIN may also be used to specify the action to be taken by a **FOR...NEXT** loop.

This operator is different from **<=** in the sense that both operands are of, or will be converted to, the **set** type.

Example:
```
10 B~ = {2,3,9} :IF 2 WITHIN B~ THEN PRINT "{2} is a subset of B~"
```

See also: **=** **<>** **<** **<=** **>** **>=** **Precedence**

XOR **Logical Exclusive OR** Operator

Purpose: Perform a logical exclusive OR on two binary integers or long integers.

Usage: *exp1* XOR *exp2*

exp1, *exp2* Any expressions, including **boolean** expressions, that produce an integer or long integer result.

The operands must be either **integer** or **long integer**. Any other operand types will cause a Type Mismatch error to occur. The result is the same type as the largest operand. If the two operands are different sizes, **integer type conversion** is performed.

MotionBASIC performs a **bitwise** XOR, according to the XOR **truth table**. See the chapter on **bitwise operations** for an example.

See also: **NOT** **AND** **OR** **Precedence**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Purpose: Determine if the "once-per-revolution" reference signal (ENCZ) of the specified axis encoder is asserted; or has been asserted since last cleared. Clear and arm an edge sensitive latch.

Usage: ***var%*** = ZREF@[(***axes***)] 'w/ multiple axes reference, ZREF@s are ORed
Syntax: ZREF@[(***axes***)] = 0

var A **Boolean** value which will be TRUE (-1) if ENCZ either is asserted, or has been asserted since last cleared, depending on **SENS.MODE@**. Otherwise it will be FALSE (0).

This input signal is used to capture axis position **POS.ZREF@**. ZREF@ can initiate a motion **AFTER** it occurs, terminate a motion waiting **UNTIL** it occurs, and it can be used in any **boolean** expression.

Setting ZREF@ to 0 always clears it if it was configured by **SENS.MODE@** as **R**ising or **F**alling edge sensitive. If configured level sensitive, it may be immediately re-asserted depending on the state of the input.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Impact of Set Variables on MotionBASIC

A. **Variables**
   1. The **variable** type character is '~'.
   2. Set **constants**, require four bytes of storage and are represented as {<elem1>,<elem2>,<elem3>, ...}. e.g. {1,16,31} is a set of three elements. Set elements must be between 1 and 32.
   3. Set **variables** are represented as variable names followed by the "set type" character. For example, set.variable~ is a set variable.
   4. **Type Conversion**
      a. SET <-> SNG not allowed
      b. SET <-> DBL not allowed
      c. SET <-> STR not allowed
      d. SET <-> INT
         1) a% = a~ will assign the value of the lowest numbered element in a~ to a%.
         2) a~ = a% will treat the value of a% as a set element and assign it to a~

      e. SET <-> LNG
         1) a& = a~ will assign the value of the lowest numbered element in a~ to a&.
         2) a~ = a& will treat the value of a& as a set element and assign it to a~
   5. Type promotion is from Integer/Long to set. For example, (3+{4}) or ({4}+3) will result in {3,4}.
   6. The **boolean** value of an empty set is FALSE otherwise it is TRUE.

B. **Operators**
   1. Arithmetic
      a. (a~ **+** b~) is equivalent to the union of a~ and b~.
      b. (a~ **-** b~) is the set of all elements in a~ that are not also elements in b~.
      c. (a~ ***** b~) is equivalent to the intersection of a~ and b~.
   2. Relational
      a. a~**=** b~ is TRUE if and only if the two sets are identical.
      b. a~**<>**b~ is TRUE if the two sets are not identical.

      c. a~**<** b~ is TRUE if a~ is a subset of b~ but not equal to b~.
      d. a~**<=**b~ is TRUE if a~ is a subset of b~.
      e. a~**>** b is TRUE if b~ is a subset of a~ but not equal to a~.
      f. a~**>=**b~ is TRUE if b~ is a subset of a~.
      g. a~ **WITHIN** b~ is the same as a~**<=**b~ except that WITHIN always convert its operands into set type first.
   3. Bitwise
      a. **NOT** a~ is equivalent to the complement of a~.

C. **Functions**
   1. **VAL**("{elem1,elem2,elem3, ...}") will return a set containing elem1, elem2, elem3, etc.
   2. **STR$**({elem1,elem2,elem3,...}) will return the string "{elem1,elem2,elem3,...}"

D. **Statements**
   1. PRINT a~ will display "elem1,elem2,elem3, ..."
   2. **DEFSET** <char1>[-<char2>]

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

3. All input statements allow set variables. For example, INPUT @, READ, INPUT, etc.
4. Added "Syntax 2" to **FOR...NEXT**

E. **Motion Statements**
MOVE, **GEAR**, and **HALT** use a **set** type axis parameter. For example, MOVE 3 FOR ... will be changed by automatic **type conversion** to MOVE {3} FOR ... The statement MOVE {1,2} FOR .... will move both axis 1 and 2.

F. **ORMEC Variables**
Axis oriented ORMEC variables are indexed on **set** variables. e.g. **AXIS.SET@** = 3 will be changed by automatic **type conversion** to **AXIS.SET@** = {3}. MODE@(3) will be interpreted as MODE@({3}).

Some ORMEC variables are **set** variables and return set values. For example, the statement AXES~ = **AXIS.SET@** assigns the current "default set of servos" to variable AXES~.

Tracing the setting of **AXIS.SET@** = {1,2} will display **AXIS.SET@** = {1,2}.

ORMEC Variables that return set type values are: **FAULT@**, **AXIS.LIST@**, **AXIS.SET@**, **AXIS.FLT1@**, **AXIS.FAULT@** and **AXIS.OTL@**

G. Set variable **Error Codes**
1. **#1224**: Expected Set

ITM Driver Support

The **ITM-270** operator interface is supported via a MotionBASIC device driver and through the use of standard statements/functions:

WIDTH, **CLOSE**, **EOF**, **INPUT #**, **PRINT #**, and **IOCTL**

The ITM Driver is turned on when an ITM device is opened.

See: **OPEN ITM**

The following documentation covers the special use of the standard statements and functions that are used with an ITM-270 device:

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

WIDTH with ITM Support

Purpose: Keep long messages from being refreshed.

Syntax: WIDTH [[#]*devnum*,] [*len*]

devnum the device number under which the ITM was OPENed

length the maximum length of a message that will be automatically refreshed on the ITM's display. Range 15 to 80. Default: 15

Example:

```
OPEN "R",1,"COM1:ITM=5"          'Open COM1 with ITM Support
WIDTH #1, 20            ' Set refresh length to 20 characters
```

NOTE: Messages longer than *length* are not refreshed. This allows the user to scroll over the entire message using the ITM's < and > keys. See: **ITM Display**

CLOSE with ITM Support

Purpose: It closes the port and turns off the ITM Driver.

Syntax: CLOSE [devnum]

devnum the device number under which the ITM was OPENed

Example:

```
OPEN "R",1,"COM1:ITM=10"    ' Open COM1 with ITM support.
...
CLOSE 1       ' Close COM1 and terminate ITM-270 support.
```

EOF with ITM Support

Purpose: Determine that there is data available at the ITM. It returns a **FALSE** flag when data at the ITM is ready.

Syntax: EOF(devnum)

devnum the device number under which the ITM was OPENed

Example:

```
IF NOT EOF(1) THEN          ' Check for ITM Data ready
   INPUT #1, DATA$,KEYNUM   ' Read ITM Data
   DATA!=VAL(DATA$)         ' Convert DATA to a value.
ENDIF
```


Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

INPUT # with ITM Support

Purpose: Extract a response from the ITM-270. The response is always a string and an integer.

Syntax: INPUT #*devnum, value$, keynum*

devnum the device number under which the ITM was OPENed

value$ contains a string representation of the value entered (or null if it was a function key only).

keynum standard **EXITKEY** values for ENTER and F1 - F6 (there are only 6 function keys. See: **ITM Function Keys**.)

The user program should check both the *value$* and the *keynum* received from the ITM-270. Operators could enter numeric data and then press a Function Key, rather than the Enter Key as one might expect.

IOCTL with ITM Support

Purpose: Use the features of an ITM-270. (e.g. lighted function keys, flashing display, etc.)

Syntax: IOCTL [#]*devnum, ctrl$*

devnum the device under which the ITM was OPENed

ctrl$ A string expression to be passed to the ITM driver.

"LED *n state*"
n is LED number or **set** specification

state ON
OFF
or number for blink rate

"DISP *flash*"
flash 0- No Change
1 - Flashing Off
2 - Flashing ON

"CURSOR *size*"
size 0 - No Change
1 - No Cursor
2 - Underscore
3 - Flashing Block

```
Examples:  IOCTL #1, "LED {6} ON"        'Turn on  LED  6
           IOCTL #1, "LED {1,3,5} OFF"   'Turn off LEDs 1, 3, & 5
           IOCTL #1, "DISP 2"            'Make the display flash
           IOCTL #1, "CURSOR 3"          'Change cursor to flashing block
```

See Also: **ITM Display & Function Keys**

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

ACL.MAX@ 134
actual 241
actual positio 241
actual velocit 279
Ad 218
Adding Data to 48
addition 126
AFAULT@ 134
AFTER 135
Ah 186
AIN1 136
AIN1@ 136
ain2 136
AIN2@ 136
AIO@ 137
alarm detectio 107
ALARM@ 137
All Functions 62
All ORMEC Vari 85
All Statements 58
amber 99
Analog I/O 39
analog input 39
Analog Input O 136
Analog Input T 136
analog velocit 27
Analog Velocit 27
AND 139
Arithmetic Fun 60
Array 69
Array Dimensio 159
Array Variable 69
arrays 69
AS 175
AS 220
ASC 139
ASCII 115
ASCII Table 115
ASEN 139
ASEN@ 139
assignment 205
AT 180
AT 180
AT 214
AT.REST@ 140
AT.SPD@ 140
ATN 140
AUTO 141
AUTOLOAD.BAS 141
Automatic Line 124
axes 141
axis 141
Axis Analog In 136
Axis Analog In 136
axis control l 23
axis control m 212
Axis Encoder 31
Axis Encoder O 108
Axis Fault 106
axis fault cod 134
axis faults 134
axis modes 212
Axis Module Fa 110
Axis Operating 83
axis position 241
Axis Position 79
Axis Range Var 83
axis variables 78
AXIS.FAULT@ 142
AXIS.FLT1@ 142
AXIS.LIST@ 142
AXIS.OTL@ 143
AXIS.SET@ 143
AXIS.VAR@ 144
backspace 51
Battery Failur 105
Baud Rate 145
BAUD@ 145
BEL 117
BIN$ 145
Bitwise 65
Bitwise operat 65
block 188
Boolean 68
Boolean Axis S 85
BPS 45
Brain Board 40
BREAK 145
BS 117
BSEN 146
BSEN@ 146
BY 180
BY 180
BY 181
BY 182
BY 186
BY 214
BYE 147
Cannot Continu 96
Carriage Retur 117
Cascaded Contr 36
CHAIN 147
Chart Rules 214
CHDIR 148
Checksum Error 111
CHR$ 148
CLEAR 149
CLG 149
CLOSE 149
CLREOL 150
CLS 150
CNT.REV@ 150
COLOR 151
Command Buffer 108
Command Oversp 107
commanded 242
commanded posi 242
Commands 54
comment 122
comments 122
complement 221
concatenation 126
constant 66
Constant Gear 162
Constants 66
CONT 152
control c 96
control stack 264
Control Struct 56
control struct 56
Controller Fau 98
Controller Fau 98
Controller Fau 104
Controller Fau 104
Conversion Fun 61
COS 152
CR 117
Creating Rando 49
Creating Seque 47
CSRLIN 152
ctrl c 96
ctrl-c 96
Ctrl-C detecte 96
Current Electr 184
CW.FWD@ 153
Da 218
DATA 153
Data File 46
Data Files 46
Data type 67
data types 67
DATE$ 153
DCL.ERR@ 154
DCL.LAST@ 154
DCL.MAX@ 154
Dd 218
DEF 155
DEF FN 155
DEF SEG 155
DEFDBL 156
DEFINE 124
DEFINT 156
DEFLNG 156
DEFSET 157
DEFSNG 157
DEFSTR 157
DELAY.DIST@ 158
DELAY@ 158

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

DELETE 158
Development Co 114
device open 229
Dh 186
diagnostics 99
Differences 4
Differences fr 4
DIM 159
DIO@ 159
direct 7
direct mode 7
Direct Mode Op 7
Direct Stateme 93
directive 50
directives 50
Discrete & Ana 39
Discrete I/O 39
disk 51
disk comment 123
disk comments 123
disks 51
Display/Keyboa 56
dist 218
DIST.BIAS@ 161
DIST.GEAR@ 162
DIST.GRAT@ 162
DIST.MOVE@ 162
distance 218
Distributed Fe 27
distributed fe 27
divide 132
divide 132
division 132
division 132
Dos file-namin 289
Dos file-namin 289
Double 75
Double precisi 75
double-quotes 123
drive command 270
DRV.CMD@ 270
DRV.MAX@ 271
DSP 21
DSP Axis Modul 21
DSP Axis Modul 21
DSP Axis Modul 22
DSP Axis Modul 110
DSP I/O Termin 21
DSP I/O Termin 21
DSP Software F 109
DSP.CTR 163
DSP.CTR@ 163
DSP.DONE@ 164
DUMP 164
Duplicate Defi 92
E-STOP 106
E-Stop Input O 106
E-STOP OK 106
EBC 32
EDIT 165
Edit Keys 10
EIO.MODE@ 165
Electronic Gea 32
Electronic Lim 38
Electronic Lim 38
Electronic Lim 165
Electronic Lim 166
Electronic Lim 166
Electronic Lim 167
Electronic Lim 167
Electronic Lim 168
ELS 38
ELS1.HIGH@ 166
ELS1.LOW@ 166
ELS1@ 165
ELS2.HIGH@ 167
ELS2.LOW@ 167
ELS2@ 166
ELS3.HIGH@ 168
ELS3.LOW@ 168
ELS3@ 167
ELSE 188
ELSEIF 188
Emergency Stop 106
Encoder Backup 32
Encoder Ch-A O 103
Encoder Ch-B O 103
encoder oversp 108
encryption 45
encryption 259
END 168
ENDIF 188
EOF 169
equal to 130
ERASE 169
ERL 170
ERR 170
ERR$ 170
ERROR 171
error code 88
Error Codes 88
error handler 89
Errors 88
ESC 117
ESTOP.OK@ 171
EVENT 171
EVENT INPUT 171
Event Key 224
Event Key Trap 201
Event Keys 224
Event Processi 41
event trapping 224
Events 41
Exit Keys 173
EXITKEY 173
EXP 174
exponentiation 133
expression 10
Expression Err 92
Expression Err 92
expressions 10
Extended I/O C 40
extended key t 17
External I/O S 57
F1 51
F2 51
F3 51
F3=SEARCH 52
F4 51
Fa 183
FALSE 68
fault action 98
FAULT@ 174
Faults 98
Fd 183
Fdist 183
FIELD 175
file open 230
filename 289
filename exten 289
FILES 176
files data 46
files MotionBA 44
files program 44
files random a 48
files random a 49
files random a 49
files sequenti 46
files sequenti 47
files sequenti 47
files sequenti 48
FIX 177
flash 20
FN 155
Fn code 118
Fncode 200
Follower 36
FOR 177
FOR 180
FOR 214
FOR...NEXT 177
FORMAT 179
forward hardwa 187
Forward Overtr 231
Forward overtr 231
forward softwa 264
FRE 179
Full-duplex 114

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

function 60
function key 51
function key t 17
Function Keys 15
Function Keys 15
Functions 60
Fwd hardware T 187
Fwd software T 264
Garbage Collec 77
GEAR 180
GEAR AT 182
GEAR Chart 180
Gear Distance 162
GEAR FOR 181
Gear Ratio 32
Gear Ratio Mul 82
Gear Speed Mul 82
GEAR Statement 180
GEAR Syntax Ch 180
GEAR.RATIO@ 184
geared 32
Generation III 219
GET 184
GOSUB 185
GOTO 185
greater than 131
greater than o 131
green 99
GWBASIC 4
HALT 186
HALT Chart 186
HALT Statement 186
HALT Syntax Ch 186
Hardware Hands 114
hardware multi 35
Hardware Overt 38
Hardware Overt 38
HEX$ 187
Hi Axis Loop r 208
Hi Pacer Loop 208
HLF 187
HLR 187
HT 117
HTL.FWD@ 187
HTL.REV@ 187
Hypertext Func 51
Hypertext Keys 51
I/O 39
I/O Errors 92
I/O Terminatio 21
I/O Variables 78
IF 188
IF...THEN...EL 188
IFDEF 125
IFNDEF 125
Illegal Direct 96
Illegal Indire 97
Illegal While 95
impact 284
Impact of Set 284
implied GOSUB 6
IN 180
IN 180
IN 181
IN 182
IN 186
IN 214
IN.ACL@ 190
IN.DCL@ 190
IN.MOT@ 190
IN.POS@ 190
Include File D 125
Incompatible M 110
Index 1
INDEX 215
indirect 8
indirect mode 8
Indirect Mode 8
INERTIA@ 190
INKEY$ 191
INP 192
INPUT 192
INPUT # 193
INPUT @ 194
input @ using 274
input using 274
INPUT$ 196
Input/Output F 61
INPUT@ 194
input@ using 274
INPW 196
INSPD.MUL@ 234
inspeed 183
INSTR 196
INT 197
Integer 73
integer divisi 132
integer expres 10
integer sign e 71
integer type c 71
integers 73
Interface 35
Internal Error 109
INTEXPR 10
Introduction 1
Invalid Unit I 105
IO.MODE@ 198
IOCTL 197
ITM 14
ITM Display 16
ITM Display & 16
ITM Driver Sup 286
ITM Function K 16
itm open 228
ITM-270 14
ITM-270 16
Ka 26
Kaf 24
KAF@ 199
KEY 200
Key (201
KEY (n) 201
Key Event 224
Key Event Trap 201
Key Table 118
Keyboard Chara 191
Keyboard scanc 119
Keyword 120
Keywords 120
KILL 202
Kp 24
KP@ 202
Kpi 25
KPI@ 203
Kv 25
Kvf 24
KVF@ 203
Kvh 26
KVH@ 203
Kvha 27
KVHA@ 204
Kvi 25
KVI@ 204
Kvl 31
label 6
labels 6
Language Exten 20
Language Exten 20
LCASE$ 204
LEDs 99
LEFT$ 205
LEM 20
LEN 205
less than 127
less than or E 128
LET 205
LF 117
lights 99
limit 231
limit switch 37
limit switch 38
limit switch 38
limits 187
limits 187
limits 264
limits 265
line 9
Line Editor 9

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Line Format 9
LIST 205
literals 66
LNGEXPR 10
LOAD 206
LOC 206
LOCATE 207
LOF 207
log 149
LOG 208
logical operat 65
Long 74
Long Integer 74
Loop Architect 23
loop output 270
loop rate 208
loop update 208
LOOP.RATE@ 208
Loss of Feedba 103
LSET 208
Machine Fault 90
Machine I/O Co 198
maximum loop o 271
maximum output 271
maximum positi 237
MBVER$ 209
MD.MODE@ 209
memory 22
memory 156
Memory Card 19
Memory Cards 19
Memory Conserv 18
Memory Disk 20
Memory Disks 20
Memory Dump 109
Memory Fault 100
memory map 18
Memory Usage 72
Memory Usage f 72
MERGE 210
Microsoft BASI 4
MID$ 210
minus 127
Miscellaneous 58
Missing Motion 108
MKDIR 211
MMI 14
MMI Display 15
MMI Function K 15
MMI function k 17
MMI Keypad Lay 16
mmi open 227
MMI-840 14
MMI-840 Functi 15
MOD 211
mode 212
MODE@ 212
Model 20 13
Model 40 12
Modifiers 55
Modulo Positio 39
Motion Command 23
Motion Control 11
Motion Control 11
motion error 94
Motion Errors 94
Motion queue 23
Motion Stateme 55
Motion Stateme 55
Motion Stateme 55
MotionBASIC 2
MotionBASIC Ev 41
MotionBASIC Fi 44
MotionBASIC Fi 44
MotionBASIC Li 10
MotionBASIC Ov 2
MotionBASIC Po 7
MotionBASIC Sy 53
MotionCARD 20
MotionCARD Fau 111
MotionCARDs 20
MotionDATA 34
MotionDATA Act 102
MotionDATA Err 108
MotionDATA Int 35
motiondata mon 209
MotionDATA Not 97
MotionDATA Ope 209
MotionNET 104
MotionNET Alar 102
MotionNET Alar 102
MotionNET Talk 104
MotionPRO 41
MotionPRO dire 50
MotionPRO Dire 50
MOVE 214
MOVE AT 217
MOVE Chart 214
MOVE FOR 215
MOVE STATEMENT 213
MOVE STATEMENT 213
MOVE Syntax Ch 214
MOVE TO 216
MP.CONFIG 219
MTR.ACL.LIM@ 219
MTR.SPD.LIM@ 220
multiplication 126
NAME 220
negation 127
NEW 221
NEW RESTORE 221
NEXT 177
No Fault 113
No Fault LED 113
No Fault Relay 113
no MotionDATA 108
Non-Printing A 117
non-volatile m 18
normalization 29
Normalization 29
NOT 221
Not enough mem 111
not equal 129
Notation 53
Numeric Axis S 84
Numeric Expres 10
NUMEXPR 10
OCT$ 222
OFF 68
OIT 14
OIT Display 15
OIT Function K 15
oit open 226
OK to Transmit 101
ON 68
ON ERROR 222
ON EVENT 224
ON EVENT...GOS 224
ON KEY 224
ON KEY...GOSUB 224
ON...GOSUB 225
ON...GOTO 225
OPEN 226
OPEN DEVICE 229
OPEN FILE 230
OPEN ITM 228
OPEN ITM-270 228
OPEN MMI 227
open mmi-840 227
OPEN OIT 226
open wire 103
Operator Inter 14
Operator Inter 14
Operators 64
OR 231
ORMEC Service 113
ORMEC Variable 78
ORMEC Variable 78
OTL.FWD@ 231
OTL.REV@ 232
OUT 233
Out of Memory 95
output mode 212
Output Select 209
OUTSPD.MUL@ 234
outspeed 183
OUTW 234
overtravel 37

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

overtravel 38
overtravel 187
overtravel 187
overtravel 231
overtravel 264
overtravel 265
Overtravel Lim 37
Overtravel Lim 37
Overtravel lim 37
Overtravel Lim 232
Pa 183
Pacer 35
Pacer Overspee 108
Pacer Position 82
Pamux 40
parameters 183
pass-thru 35
path 289
PB1 234
PB1@ 234
PB1LOAD.BAS 235
PCR.SPD.LIM@ 235
PCT.REV@ 235
Pd 183
Pdist 183
PEEK 236
PEEKW 236
PERR.INPOS@ 237
PERR.MAX@ 237
PERR.NORM@ 238
PG.LEN@ 238
pi 79
plus 126
POKE 239
POKEW 239
poll 240
POS 240
POS.ABS@ 240
POS.ACT@ 241
POS.ASEN@ 241
POS.BSEN@ 242
POS.CMD@ 242
POS.DELAY@ 243
POS.DIV@ 245
POS.ERR@ 243
POS.MOD@ 244
POS.MUL@ 245
POS.ZREF@ 246
position 218
position 241
Position Comma 107
Position Comma 161
position comma 242
position error 237
Position Loop 24
Position Loop 25
position mode 212
Position Modul 244
Power Fault 100
Power Fault - 105
Power Fault-Te 112
powerup 7
powerup operat 7
PPS.DIV@ 246
PPS.MUL@ 246
Precedence 64
PRINT 247
PRINT # 248
print # using 274
PRINT @ 247
print using 274
print# 248
print# using 274
PRINT@ 247
print@ using 274
Program Encryp 45
Program Error 100
program errors 100
Program file 44
Program Files 44
program initia 8
Program Initia 57
Program Labels 6
program line 9
Program Struct 91
Program Struct 91
PUT 249
PW@ 249
RAM Checksum E 104
Random Access 48
Random Access 48
RANDOMIZE 250
READ 251
Ready to Recei 101
Receiving Moti 102
Receiving Moti 104
Receiving Seri 101
red 99
REM 251
RENUM 252
RENUM Chart 252
RENUM Statemen 252
RENUM Syntax C 252
REPEAT 253
Reserved Words 120
RESTORE 254
Restore Non-Vo 221
RESUME 255
RETURN 256
rev hardware T 187
rev software T 265
reverse hardwa 187
reverse softwa 265
RIGHT$ 256
RMDIR 257
RND 257
RSET 258
Rules 214
RUN 258
SAVE 259
Scan Code 119
Scan Codes 119
scancode 119
scancodes 119
SCURVE@ 260
search 52
SEG 155
segment 156
segmented memo 156
segments 156
Select 209
Sending Motion 102
Sending Motion 104
Sending Serial 101
SENS.MODE@ 261
Sequential Fil 46
Sequential Fil 46
Serial Port St 101
Servo Control 23
Servo Fault 101
Servo Fault 106
Servo Loop Tun 84
servo loop upd 208
servo loop upd 208
Servodrive Ala 107
servodrive ala 137
servodrive ala 137
Set 72
set complement 221
set difference 127
set expression 10
set intersecti 126
set membership 127
set membership 128
set membership 129
set membership 130
set membership 131
set membership 131
set membership 282
set union 126
set variables 284
SETEXPR 10
setup 83
setup 83
Setup 84
Setup 85
SGN 262
shared memory 22

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

shared memory 22
sign extension 71
signal loss 103
signal loss de 103
SIN 262
Single 74
Single precisi 74
single-quote 122
Software Overt 37
Software Overt 37
software overt 264
software overt 265
SPACE$ 263
SPC 263
SPD.MAX@ 263
speed 218
speed units 80
SQR 264
sram 20
SRL1 145
STACK 264
stack clear 264
statement 55
Statements 55
status 99
STEP 177
STL.FWD@ 264
STL.REV@ 265
STOP 265
STR$ 265
String 76
String Functio 60
String Garbage 77
String Space C 110
String Space F 110
string space s 149
String Variabl 76
STRING$ 266
Strings 76
Structured Pro 6
subroutine 185
subroutines 185
subtraction 127
Syntax 52
Syntax 53
syntax 1 52
syntax 2 52
syntax 3 52
syntax 4 52
syntax 5 52
Syntax Chart R 214
Syntax Error 90
Syntax Errors 90
SYSTEM 266
system error 95
System Errors 95
System Functio 61
System Status 99
System Watchdo 103
Ta 218
TAB 266
TALK 267
TAN 267
Td 218
Temporary Powe 112
Terminations 21
Text Editor 50
Th 186
THEN 188
time 218
TIME$ 268
TIME.MUL@ 268
TIMER 269
TO 177
TO 214
Token Buffer O 97
Torque Mode Se 23
torque mode se 28
Torque Mode Se 28
TRACE 269
TRACE VAL 270
Troubleshootin 113
Troubleshootin 113
TRQ.CMD@ 270
TRQ.GAIN@ 271
TRQ.MAX@ 271
TRUE 68
truth table 65
Truth Tables 65
two's compleme 71
Type Conversio 70
Typical 54
UCASE$ 272
UNDEF 126
union 126
Unit Fault 174
Unit Fault Cod 174
Unit Identific 272
Unit Parameter 78
Unit Parameter 78
UNIT.ID@ 272
UNTIL 272
usage 52
usage 1 52
usage 2 52
usage 3 52
usage 4 52
User 1 101
User 2 101
User Accelerat 81
User Defined E 90
User Defined F 90
User Functions 155
User Speed Uni 80
User Trappable 89
User Trappable 89
User Trappable 89
user trapped 89
User Units 79
USER1.LED@ 273
USER2.LED@ 274
USING 274
USING 2 214
USR.ACL.LIM@ 278
USR.SPD.LIM@ 278
VAL 279
var 63
variable 67
variable name 67
Variable Names 67
Variables 67
VEL.ACT@ 279
VEL.CMD@ 279
VEL.GAIN@ 31
VEL.LAST@ 280
velocity 218
Velocity Feedf 24
Velocity Loop 25
Velocity Loop 25
Velocity Loop 31
velocity mode 30
velocity mode 212
Velocity Mode 30
velocity mode 30
Velocity Mode 30
velocity mode 30
velocity mode 30
Velocity Mode 31
Velocity Obser 26
Velocity Obser 28
Velocity Obser 203
Velocity out o 94
VLTC@ 280
WAIT 280
Watchdog Timeo 112
Watchdog Timer 103
WEND 281
WHILE 281
WHILE...WEND 281
WIDTH 282
WITHIN 282
Word 10
write enable 249
write protect 249
XOR 283
ZREF 283
ZREF@ 283
{ 72

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Errors Generated w/ ITM Support

The only ITM specific errors are ones that deal with a breakdown of communications between the controller and the ITM-270.

In the event of a communications break down, a flag is set internal to the device driver. The next time the device is accessed via an **EOF**, **IOCTL**, **INPUT #** or **PRINT #** command, an error message will be generated.

Statement	Error	
PRINT #	**#1310**	Device Write Failed
IOCTL	**#1310**	Device Write Failed
INPUT #	**#1312**	Device Read Failed
EOF	**#1312**	Device Read Failed

DOS File-Naming Conventions

When using **MotionBASIC Files** a unique name is assigned to each file.
A file's name appears as follows: **FILENAME.EXT**

A file's name has two parts, the filename and an extension.
A period separates a filename from an extension.

A filename can be from 1 to 8 characters long.
An extension is 1 to 3 characters long and indicates the type of file.
Only letters, numbers and _ (underscore) are valid filenaming characters.

In MotionBASIC, **program files** should have the extension, **BAS**

The following filenames have special significance:

AUTOLOAD.BAS - Loads from a **Memory Disk** at each **powerup**.
PB1LOAD.BAS - Loads from a **Memory Disk** at powerup when **PB1** is held.
SETUP.BAS - The test program provided with controllers & w/ **MotionPRO**.
CONFIG.BAS - Recommended name for the file storing the **MP.CONFIG** routine.

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

MotionBASIC p 8
! 74
!AUTO 124
!INCLUDE 125
75
#1000 90
#1001 90
#1002 90
#1003 90
#1004 90
#1005 90
#1006 90
#1007 90
#1008 90
#1009 90
#1010 90
#1011 90
#1012 90
#1014 90
#1015 90
#1101 91
#1102 91
#1103 91
#1104 91
#1105 91
#1106 91
#1107 91
#1108 91
#1109 91
#1110 91
#1111 91
#1112 91
#1114 92
#1201 92
#1202 92
#1203 92
#1204 92
#1205 92
#1206 92
#1207 92
#1208 92
#1209 92
#1210 92
#1211 92
#1212 92
#1213 92
#1214 92
#1301 92
#1302 92
#1303 92
#1304 92
#1305 93
#1306 92
#1307 92
#1308 92
#1309 92
#1310 92
#1311 92
#1312 92
#1313 92
#1314 92
#1315 92
#1316 92
#1317 92
#1318 92
#1319 92
#1601 94
#1602 94
#1604 94
#1605 94
#1606 94
#1607 94
#1608 94
#1609 94
#1610 95
#1612 94
#1614 94
#1615 94
#1616 94
#1801 95
#1802 96
#1803 95
#1804 96
#1805 96
#1806 95
#1807 95
#1808 97
#1810 95
#1811 97
#1812 95
#1813 95
#1814 95
#1815 97
#1816 95
#1900 90
#1901 90
#1902 90
#1903 90
#1904 90
#1905 90
#1906 90
#1907 90
#1908 90
#1909 90
#1910 106
#1911 106
#1912 109
#1913 110
#1914 110
#1915 110
#1916 111
#1917 111
#1918 111
#1919 111
#1998 112
#1999 112
$ 76
% 73
& 74
' 122
'! 123
'! 124
'! 124
'! 125
'! 125
'! 126
'!AUTO 124
'!DEFINE 124
'!ELSE 125
'!ENDIF 125
'!IFDEF 125
'!IFNDEF 125
'!INCLUDE 125
'!UNDEF 126
'' 123
* 126
+ 126
- 127
/ 132
10 107
< 127
<= 128
<> 129
= 130
> 131
>= 131
? 247
@ 78
\ 132
^ 133
Aa 218
abort 8
ABS 133
abs encoder 240
absolute encod 240
absolute encod 240
absolute encod 240
absolute posit 240
absolute value 133
accel 218
Accel or Decel 94
Acceleration F 24
Acceleration G 26
acceleration u 81
Accessing Rand 49
Accessing Sequ 47
ACL.CMD@ 133
ACL.LAST@ 133

Artisan Scientific - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisan-scientific.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP