

---

## A Acceleration

|          |   |         |     |
|----------|---|---------|-----|
| Type     | Motion  | Product | Rev |
| Syntax   | <!><@><a>A<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>                   | 6K      | 5.0 |
| Units    | r = units/sec/sec   |         |     |
| Range    | 0.00001 - 39,999,998 (depending on the SCLA scaling factor) |         |     |
| Default  | 10.0000   |         |     |
| Response | A: *A10.0000,10.0000,10.0000,10.0000 ...<br>1A: *A10.0000   |         |     |
| See Also | [ A ], AA, AD, ADA, DRES, ERES, GO, MC, SCALE, SCLA, TSTAT  |         |     |

---

The Acceleration (A) command specifies the acceleration rate to be used upon executing the next go (GO) command.

**UNITS OF MEASURE and SCALING:** refer to page 16.

The acceleration remains set until you change it with a subsequent acceleration command. Accelerations outside the valid range are flagged as an error, with a message \*INVALID DATA-FIELD x, where x is the field number. When an invalid acceleration is entered the previous acceleration value is retained.

If the Deceleration (AD) command has not been entered, the acceleration (A) command will set the deceleration rate. Once the deceleration (AD) command has been entered, the acceleration (A) command no longer affects deceleration.

**ON-THE-FLY CHANGES:** You can change acceleration *on the fly* (while motion is in progress) in two ways. One way is to send an immediate acceleration command (!A) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered acceleration command (A) followed by a buffered go command (GO).

**Example:**

```
SCALE1                ; Enable scaling
SCLA25000,25000,1,1   ; Set the acceleration scaling factor for axes 1 & 2 to
                      ; 25000 steps/unit, axes 3 & 4 to 1 step/unit
SCLV25000,25000,1,1   ; Set the velocity scaling factor for axes 1 & 2 to
                      ; 25000 steps/unit, axes 3 & 4 to 1 step/unit
@SCLD1                ; Set the distance scaling factor for all axes to
                      ; 1 step/unit
DEL proga             ; Delete program called proga
DEF proga             ; Begin definition of program called proga
MA0000                ; Incremental index mode for all axes
MC0000                ; Preset index mode for all axes
A10,12,1,2            ; Set the acceleration to 10, 12, 1, & 2 units/sec/sec
                      ; for axes 1, 2, 3 & 4
V1,1,1,2             ; Set the velocity to 1, 1, 1, & 2 units/sec for
                      ; axes 1, 2, 3 & 4, respectively
D100000,1000,10,100  ; Set the distance to 100000, 1000, 10, & 100 units for
                      ; axes 1, 2, 3 & 4
GO1100                ; Initiate motion on axes 1 and 2, 3 and 4 do not move
END                   ; End definition of program called proga
```

---

## [ A ] Acceleration Assignment

|          |  |                |            |
|----------|--|----------------|------------|
| Type     | Assignment or Comparison                               | <b>Product</b> | <b>Rev</b> |
| Syntax   | See below  | 6K             | 5.0        |
| Units    | units/sec/sec  |                |            |
| Range    | 0.00001 - 39,999,998 (depending on the scaling factor) |                |            |
| Default  | n/a  |                |            |
| Response | n/a  |                |            |
| See Also | A, AA, AD, ADA, DRES, ERES, GO, SCALE, SCLA            |                |            |

---

The acceleration assignment command is used to compare the programmed acceleration value to another value or variable, or to assign the current programmed acceleration to a variable.

**Syntax:** VARn=aA, where n is the variable number, and a is the axis number, or A can be used in an expression such as IF(1A<25000). When assigning the acceleration value to a variable, an axis specifier must always precede the assignment (A) command or it defaults to axis 1 (e.g., VAR1=1A). When making a comparison to the programmed acceleration, an axis specifier must also be used (e.g., IF(1A<20000)). The (A) value used in any comparison, or in any assignment statement is the programmed (A) value.

**UNITS OF MEASURE and SCALING:** refer to page 16.

**Example:**

```
IF(2A<25000)      ; If the acceleration on axis 2 is less than 25000 units/sec/sec,  
                  ; then do the statements between the IF and NIF  
VAR1=2A*2        ; Variable 1 = acceleration of axis 2 times 2  
A,(VAR1)         ; Set the acceleration on axis 2 to the value of variable 1  
NIF              ; End the IF statement
```

---

## AA Average Acceleration

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Motion (S-Curve)  | <b>Product</b> | <b>Rev</b> |
| Syntax   | <!><@><a>AA<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>                  | 6K             | 5.0        |
| Units    | r = units/sec/sec   |                |            |
| Range    | 0.00001 - 39,999,998 (depending on the scaling factor)      |                |            |
| Default  | 10.00 (trapezoidal profiling is default, where AA tracks A) |                |            |
| Response | AA: *AA10.0000,10.0000,10.0000,10.0000<br>1AA: *1AA10.0000  |                |            |
| See Also | A, AD, ADA, SCALE, SCLA                                     |                |            |

---

The Average Acceleration (AA) command allows you to specify the average acceleration for an S-curve motion profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. Refer to page 13 for details on S-curve profiling.

Scaling affects the average acceleration (AA) the same as it does for the maximum acceleration (A). Refer to page 16 for details on scaling.

**ON-THE-FLY CHANGES:** You can change acceleration *on the fly* (while motion is in progress) in two ways. One way is to send an immediate acceleration command (!AA) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered acceleration command (AA) followed by a buffered go command (GO).

**Example:**

```
; In this example, axis 1 executes a pure S-curve and takes 1 second  
; to reach a velocity of 5 rps; axis 2 executes a trapezoidal profile  
; and takes 0.5 seconds to reach a velocity of 5 rps.  
SCALE0          ; Disable scaling  
DEL proga       ; Delete program called proga  
DEF proga       ; Begin definition of program called proga  
@MA0           ; Select incremental positioning mode  
@D40000        ; Set distances to 40,000 positive-direction steps  
A10,10         ; Set max. accel to 10 rev/sec/sec (axes 1 and 2)
```

```

AA5,10      ; Set avg. accel to 5 rev/sec/sec on axis 1,
            ; and 10 rev/sec/sec on axis 2
AD10,10     ; Set max. decel to 10 rev/sec/sec (axes 1 and 2)
ADA5,10     ; Set avg. decel to 5 rev/sec/sec on axis 1,
            ; and 10 rev/sec/sec on axis 2
V5,5        ; Set velocity to 5 rps on axes 1 and 2
GO11        ; Execute motion on axes 1 and 2
END         ; End definition of program called proga

```

---

## AD

### Deceleration

| Type     | Motion  | Product | Rev |
|----------|---|---------|-----|
| Syntax   | <!><@><a>AD<r>,<r>,<r>,<r>,<r>,<r>,<r>,<r>                    | 6K      | 5.0 |
| Units    | r = units/sec/sec   |         |     |
| Range    | 0.00001 - 39,999,998 (depending on the scaling factor)        |         |     |
| Default  | 10.0000 (AD tracks A)   |         |     |
| Response | AD: *AD10.0000,10.0000,10.0000,10.0000 ...<br>LAD: *AD10.0000 |         |     |
| See Also | [ A ], A, AA, ADA, DRES, ERES, GO, MC, SCALE, SCLA, TSTAT     |         |     |

---

The Deceleration (AD) command specifies the deceleration rate to be used upon executing the next go (GO) command.

**UNITS OF MEASURE and SCALING:** refer to page 16.

The deceleration remains set until you change it with a subsequent deceleration command. Decelerations outside the valid range are flagged as an error, with a message \*INVALID DATA-FIELD x, where x is the field number. When an invalid deceleration is entered the previous deceleration value is retained.

If the deceleration (AD) command has not been entered, the acceleration (A) command will set the deceleration rate. Once the deceleration (AD) command has been entered, the acceleration (A) command no longer affects deceleration. If the AD command is set to zero (ADØ), then the deceleration will once again track whatever the A command is set to.

**ON-THE-FLY CHANGES:** You can change deceleration *on the fly* (while motion is in progress) in two ways. One way is to send an immediate deceleration command (!AD) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered deceleration command (AD) followed by a buffered go command (GO).

**Example:**

```

SCALE1      ; Enable scaling
SCLA25000,25000,1,1 ; Set the acceleration scaling factor for axes 1 and 2 to
                ; 25000 steps/unit, axes 3 and 4 to 1 step/unit
SCLV25000,25000,1,1 ; Set the velocity scaling factor for axes 1 and 2 to
                ; 25000 steps/unit, axes 3 and 4 to 1 step/unit
@SCLD1      ; Set the distance scaling factor for all axes to 1 step/unit
DEL proga   ; Delete program called proga
DEF proga   ; Begin definition of program called proga
MA0000     ; Incremental index mode for all axes
MC0000     ; Preset index mode for all axes
A10,12,1,2 ; Set the acceleration to 10, 12, 1, and 2 units/sec/sec
                ; for axes 1, 2, 3 and 4, respectively
AD1,1,1,2  ; Set the deceleration to 1, 1, 1, and 2 units/sec/sec for
                ; axes 1, 2, 3 and 4, respectively
V1,1,1,2   ; Set the velocity to 1, 1, 1, and 2 units/sec for axes
                ; 1, 2, 3 and 4, respectively
D100000,1000,10,100 ; Set the distance to 100000, 1000, 10, and 100 units for
                ; axes 1, 2, 3 and 4, respectively
GO1100     ; Initiate motion on axes 1 and 2, 3 and 4 do not move
END        ; End definition of program called proga

```

---

## [ AD ] Deceleration Assignment

|          |  |                |            |
|----------|--|----------------|------------|
| Type     | Assignment or Comparison                               | <b>Product</b> | <b>Rev</b> |
| Syntax   | See below  | 6K             | 5.0        |
| Units    | units/sec/sec  |                |            |
| Range    | 0.00001 - 39,999,998 (depending on the scaling factor) |                |            |
| Default  | n/a  |                |            |
| Response | n/a  |                |            |
| See Also | [A], A, AA, AD, ADA, DRES, ERES, GO, SCALE, SCLA       |                |            |

---

The deceleration assignment command is used to compare the programmed deceleration value to another value or variable, or to assign the current programmed deceleration to a variable.

**Syntax:** VARn=aAD where n is the variable number, and a is the axis number, or [AD] can be used in an expression such as IF(1AD<25000). When assigning the deceleration value to a variable, an axis specifier must always precede the assignment (AD)command or it defaults to axis 1 (e.g., VAR1=1AD). When making a comparison to the programmed deceleration, an axis specifier must also be used (e.g., IF(1AD<20000)). The (AD) value used in any comparison, or in any assignment statement is the programmed (AD) value.

**UNITS OF MEASURE and SCALING:** refer to page 16.

**Example:**

```
IF(2AD<25000) ; If the deceleration on axis 2 is less than 25000 units/sec/sec,
                ; then do the statements between the IF and NIF
VAR1=2AD*2    ; Variable 1 = deceleration of axis 2 times 2
AD,(VAR1)    ; Set the deceleration on axis 2 to the value of variable 1
NIF          ; End the IF statement
```

---

## ADA Average Deceleration

|          |  |                |            |
|----------|--|----------------|------------|
| Type     | Motion (S-Curve)   | <b>Product</b> | <b>Rev</b> |
| Syntax   | <!><@><a>ADA<r>, <r>, <r>, <r>, <r>, <r>, <r>, <r>                 | 6K             | 5.0        |
| Units    | r = units/sec/sec  |                |            |
| Range    | 0.00001 - 39,999,998 (depending on the scaling factor)             |                |            |
| Default  | 10.00 (ADA tracks AA)  |                |            |
| Response | ADA: *ADA10.0000,10.0000,10.0000,10.0000 ...<br>1ADA: *1ADA10.0000 |                |            |
| See Also | A, AA, AD, SCALE, SCLA   |                |            |

---

The Average Deceleration (ADA) command allows you to specify the average deceleration for an S-curve motion profile. S-curve profiling provides smoother motion control by reducing the rate of change in acceleration and deceleration; this accel/decel rate of change is known as *jerk*. Refer to page 13 for details on S-curve profiling.

Scaling affects the average acceleration (AA) the same as it does for the maximum acceleration (A). Refer to page 16 for details on scaling.

**ON-THE-FLY CHANGES:** You can change deceleration *on the fly* (while motion is in progress) in two ways. One way is to send an immediate deceleration command (!ADA) followed by an immediate go command (!GO). The other way is to enable the continuous command execution mode (COMEXC1) and execute a buffered deceleration command (ADA) followed by a buffered go command (GO).

In the example below, axis 1 executes a pure S-curve and takes 1 second to return to zero velocity; axis 2 executes a trapezoidal profile and takes 0.5 seconds to return to zero velocity.

**Example:**

```
SCALE0        ; Disable scaling
DEL proga     ; Delete program called proga
DEF proga     ; Begin definition of program called proga
@MA0         ; Select incremental positioning mode
@D40000      ; Set distances to 40,000 positive-direction steps
```

```

A10,10      ; Set max. accel to 10 rev/sec/sec (axes 1 and 2)
AA5,10      ; Set avg. accel to 5 rev/sec/sec on axis 1,
            ; and 10 rev/sec/sec on axis 2
AD10,10     ; Set max. decel to 10 rev/sec/sec (axes 1 and 2)
ADA5,10     ; Set avg. decel to 5 rev/sec/sec on axis 1,
            ; and 10 rev/sec/sec on axis 2
V5,5        ; Set velocity to 5 rps on axes 1 and 2
GO11        ; Execute motion on axes 1 and 2
END          ; End definition of program

```

---

## ADDR Multiple Unit Auto-Address

|          |                          |         |     |
|----------|--------------------------|---------|-----|
| Type     | Controller Configuration | Product | Rev |
| Syntax   | <!>ADDR<i>               | 6K      | 5.0 |
| Units    | i = axis number          |         |     |
| Range    | 0 to 99                  |         |     |
| Default  | 0                        |         |     |
| Response | ADDR: *ADDR0             |         |     |
| See Also | BAUD, E, PORT            |         |     |

---

The ADDR command automatically configures unit addresses for a daisy-chain or multi-drop. This command allows up to 99 units on a chain to be uniquely addressed.

The ADDR value is stored in non-volatile memory.

### RS-232C Daisy Chain:

Sending ADDR*i* to the first unit in the chain sets its address to be (*i*). The first unit in turn transmits ADDR(*i* + 1) to the next unit to set its address to (*i* + 1). This continues down the daisy chain until the last unit of (*n*) daisy-chained units has its address set to (*i* + *n*).

### RS-485 Multi-Drop:

To use the ADDR command, you must address each unit individually before it is connected on the multi drop. For example, given that each product is shipped configured with address zero, you could set up a 4-unit multi-drop with the commands below, and then connect them in a multi drop:

1. Connect the unit that is to be unit #1 and transmit the Ø\_ADDR1 command to it.
2. Connect the unit that is to be unit #2 and transmit the Ø\_ADDR2 command to it.
3. Connect the unit that is to be unit #3 and transmit the Ø\_ADDR3 command to it.
4. Connect the unit that is to be unit #4 and transmit the Ø\_ADDR4 command to it.

If you need to replace a unit in the multi drop, send the Ø\_ADDR*i* command to it, where "i" is the address you wish the new unit to have.

To send a 6K command from the master unit to a specific unit in the multi-drop, prefix the command with the unit address and an underscore (e.g., 3\_OUTØ turns off output #1 on unit #3). The master unit (if it is not a 6K product) may receive data from a multi-drop unit.

For more information on controlling multiple 6K Series controllers in an RS-232 daisy-chain or RS-485 multi-drop, refer to the *Programmer's Guide*.

### **Example:**

```

ADDR1      ; Set the address of the first unit in the daisy-chain to 1

```

---

## [ AND ]

### And

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Operator (logical)                              | <b>Product</b> | <b>Rev</b> |
| Syntax   | See below                                       | 6K             | 5.0        |
| Units    | n/a   |                |            |
| Range    | n/a   |                |            |
| Default  | n/a   |                |            |
| Response | n/a   |                |            |
| See Also | IF, [ NOT ], [ OR ], REPEAT, UNTIL, WAIT, WHILE |                |            |

---

The AND command is used in conjunction with the program flow control commands (IF, REPEAT, UNTIL, WHILE, WAIT). The AND command logically links two events. If each of the two events are true, and are linked with an AND command, then the whole statement is true. This fact is best illustrated by example.

**Example 1:** IF (VAR1>0 AND VAR2<3) : TPM : NIF

If variable 1 = 1 and variable 2 = 1, then the expression within the IF statement is true, and the commands between the IF and the NIF will be executed.

**Example 2:** WHILE (VAR1=1 AND VAR2=2) : TPM : NWHILE

If variable 1 = 1 and variable 2 = 1, then the expression within the WHILE statement is false, and the commands between the WHILE and the NWHILE will not be executed.

To evaluate an expression (Expression 1 AND Expression 2 = Result) to determine if the whole expression is true, use the following rules:

TRUE AND TRUE = TRUE  
TRUE AND FALSE = FALSE  
FALSE AND TRUE = FALSE  
FALSE AND FALSE = FALSE

---

## [ ANI ]

### Analog Input Value

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Assignment or comparison                        | <b>Product</b> | <b>Rev</b> |
| Syntax   | See below                                       | 6K             | 5.0        |
| Units    | n/a   |                |            |
| Range    | n/a   |                |            |
| Default  | n/a   |                |            |
| Response | n/a   |                |            |
| See Also | ANIRNG, [ FB ], [ PANI ], SFB, TANI, TFB, TPANI |                |            |

---

Use the ANI operator to assign the voltage level present at one of the analog inputs (ANI) to a variable, or to make a comparison against another value. The ANI value is measured in volts and does not reflect the effects of distance scaling (SCLD), position offset (PSET), or commanded direction polarity (CMDDIR). To assign/compare the ANI input value, as affected by SCLD, PSET, and CMDDIR, use the PANI command or the FB command.

The ANI value is derived from the voltage applied to the corresponding analog input and ground. The analog value is determined from a 12-bit analog-to-digital converter. Under the default ANI voltage range, set with ANIRNG, the range of the ANI operator is -10.000VDC to +10.000VDC (see ANIRNG command for optional voltage ranges).

**Syntax:** VARn=<B>ANI.i where “n” is the variable number, “<B>” is the number of the I/O brick, and “i” is I/O brick address where the analog input resides; or ANI can be used in an expression such as IF(1ANI.2=2.3). If no brick identifier (<B>) is provided, it defaults to 1. To understand the I/O brick addressing convention, refer to page 6.

**Example:**

```
VAR2=3ANI.2      ; Voltage value at analog input 2 on I/O brick 3 is assigned
                  ; to variable 2
IF(1ANI.1<8.2)   ; If voltage value at analog input 1 on brick 1 < 8.2V, do the
                  ; commands between the IF statement and the NIF statement.
TREV             ; Transfer revision level
NIF              ; End if statement
```

---

## ANIEN Analog Input Enable

|          |   |         |     |
|----------|---|---------|-----|
| Type     | Inputs  | Product | Rev |
| Syntax   | To enable only: <!><B>ANIEN<.i>=<E><br>To override only: <!><B>ANIEN<.i>=<r>  | 6K      | 5.0 |
| Units    | B = I/O brick number<br>i = input location on I/O brick "B"<br>E = Enable<br>r = volts  |         |     |
| Range    | B = 1-8<br>i = 1-32 (dependent on I/O brick configuration)<br>r = -10.000 to +10.000 (voltage override value)                               |         |     |
| Default  | E (enabled)   |         |     |
| Response | 2ANIEN: *2ANIENx,x,x,x,x,x,x,x (SIM slot 1)<br>x,x,x,x,x,x,x,x (SIM slot 2)<br>E,E,E,E,E,E,E,E (SIM slot 3)<br>x,x,x,x,x,x,x,x (SIM slot 4) |         |     |
| See Also | [ ANI ], ANIFB, ANIMAS, ANIRNG, FOLMAS, TANI, TIO   |         |     |

---

The Analog Input Enable (ANIEN) command enables or disables specific analog inputs. The default state for each input is the enabled condition. ANIEN can also be used to set analog inputs to specific override voltage levels. To disable an analog input, set an override voltage of 0.

**Performance:** The rate at which the controller samples each analog input depends on how many are enabled on the SIM; each enabled analog input adds 2 ms to the sample rate for all analog inputs on the SIM. For example, if 4 of the 8 analog inputs on a SIM are enabled, the sample rate for any specific input on the same SIM is 8 ms (4 inputs x 2 ms). Disabling input channels increases the performance of the remaining channels; this is important if an input channel is to be used as a servo feedback source (ANIFB and SFB selection) or Following source (ANIMAS and FOLMAS selection).

### Example:

```
1ANIEN.9=E,E      ; Enable the 1st & 2nd analog input in SIM slot 2 (I/O
                  ; locations 9 & 10) on I/O brick 1.
1ANIEN.11=E,2.5   ; Override the 3rd analog input in SIM slot 2 (I/O location 11)
                  ; on I/O brick 1 with a voltage of 2.4 volts.
2ANIEN           ; Check status of analog inputs on I/O brick 2. As an example,
                  ; a response of *2ANIENx,x,x,x,x,x,x,x
                  ;           x,x,x,x,x,x,x,x
                  ;           E,E,E,E,E,E,E,E
                  ;           x,x,x,x,x,x,x,x
                  ; indicates that an analog input SIM is installed in slot 3 of
                  ; I/O brick 2 and all eight channels are enabled ("E").
```

---

## ANIFB Assign Analog Inputs as Axis Feedback

|          |   |         |     |
|----------|---|---------|-----|
| Type     | Controller Configuration                                    | Product | Rev |
| Syntax   | <!>ANIFB<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>,<B-i>     | 6K      | 5.0 |
| Units    | B = I/O brick number<br>i = input location on I/O brick "B" |         |     |
| Range    | B = 1-8<br>i = 1-32 (dependent on I/O brick configuration)  |         |     |
| Default  | 0-0 (No assignment)   |         |     |
| Response | ANIFB: *ANIFB1-1,1-9,0,0,0,0,0,0                            |         |     |
| See Also | [ ANI ], ANIEN, ANIRNG, SFB, TANI, TIO                      |         |     |

---

The ANIFB command determines which analog (ANI) inputs to use as feedback sources for specific axes when ANI feedback is selected by the SFB command. The ANIFB command only has an effect if ANI feedback is selected by a subsequent SFB command (ANIFB command must be issued before the SFB command).

### Example

```
ANIFB,,,,,4-17   ; Select the 1st analog input channel in SIM slot 3
                  ; (I/O location 17) of I/O brick 4 to be used as
                  ; feedback for axis 6
SFB,,,,,2        ; Select analog input feedback for axis 6
```

## ANIMAS Assign Analog Inputs to Axes

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Following   | <b>Product</b> | <b>Rev</b> |
| Syntax   | <!>ANIMAS<B-i>, <B-i>, <B-i>, <B-i>, <B-i>, <B-i>, <B-i>, <B-i> | 6K             | 5.0        |
| Units    | B = I/O brick number<br>i = input location on I/O brick "B"     |                |            |
| Range    | B = 1-8<br>i = 1-32 (dependent on I/O brick configuration)      |                |            |
| Default  | 0-0 (No assignment)   |                |            |
| Response | ANIMAS: *ANIMAS1-1,1-9,0,0,0,0,0,0                              |                |            |
| See Also | ANIEN, FOLMAS   |                |            |

The ANIMAS command assigns an analog input channel to a specific Following master axis for use when an ANI master is selected with the FOLMAS command. The ANIMAS command only has an effect if an analog input Following master is selected with a subsequent FOLMAS command (ANIMAS command must be issued before the FOLMAS command).

### Example

```
ANIMAS,,,,,4-17      ; Select the first analog input channel in SIM slot 3
                    ; (I/O location 17) of I/O brick 4 to be used for
                    ; master axis 6
FOLMAS62,62          ; Define axes 1 and 2 to be followers of the analog input
                    ; selected for master axis 6
```

## ANIRNG Analog Input Voltage Range

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Controller Configuration  | <b>Product</b> | <b>Rev</b> |
| Syntax   | <!><B>ANIRNG<.i><=i>  | 6K             | 5.0        |
| Units    | B = I/O brick number<br>1st i = input location on I/O brick "B"<br>2nd i = voltage range selector number  |                |            |
| Range    | B = 1-8<br>1st i = 1-32 (dependent on I/O brick configuration)<br>2nd i = 1 (0 to +5VDC),<br>2 (-5 to +5VDC),<br>3 (0 to +10VDC), or<br>4 (-10 to +10VDC) |                |            |
| Default  | 4 (range is set to -10 to +10VDC)   |                |            |
| Response | 2ANIRNG: *2ANIRNGx,x,x,x,x,x,x,x<br>4,4,4,4,4,4,4,4,4<br>x,x,x,x,x,x,x,x<br>x,x,x,x,x,x,x,x<br>2ANIRNG.9: *4  |                |            |
| See Also | [ ANI ], ANIEN, ANIFB, JOYCDB, JOYCTR, JOYEDB, [ PANI ], SCLA, SCLV, SFB, TANI, TIO, TPANI  |                |            |

Use the ANIRNG command to select voltage ranges for specific analog inputs on the expansion I/O brick connected to your 6K product. The default range for all analog inputs -10VDC to +10VDC.

Be aware that changing the analog input voltage range affects these settings:

| ANIRNG Setting | Voltage Range | Counts/volt resolution (see PANI & TPANI) | Calculation for minimum accel ** | Calculation for maximum accel ** | Calculation for maximum velocity ** |
|----------------|---------------|---|----------------------------------|----------------------------------|-------------------------------------|
| 1              | 0 to +5VDC    | 819                                       | $\frac{0.819}{SCLA}$             | $\frac{819,000}{SCLA}$           | $\frac{819,000}{SCLA}$              |
| 2              | -5 to +5VDC   | 410                                       | $\frac{0.410}{SCLA}$             | $\frac{410,000}{SCLA}$           | $\frac{410,000}{SCLA}$              |
| 3              | 0 to +10VDC   | 410                                       | $\frac{0.410}{SCLA}$             | $\frac{410,000}{SCLA}$           | $\frac{410,000}{SCLA}$              |
| 4              | -10 to +10VDC | 205                                       | $\frac{0.205}{SCLA}$             | $\frac{205,000}{SCLA}$           | $\frac{205,000}{SCLA}$              |

\*\* These calculations are for servo axes using an analog input as it's position feedback source (see ANIFB and SFB).

### Example

```
2ANIRNG.9=3        ; For the 1st analog input on SIM2 of I/O brick 2,
                    ; select a voltage range of 0 to +10VDC
```

---

**[ AS ]****Axis Status**

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Assignment or Comparison                                      | <b>Product</b> | <b>Rev</b> |
| Syntax   | See below   | 6K             | 5.0        |
| Units    | n/a   |                |            |
| Range    | n/a   |                |            |
| Default  | n/a   |                |            |
| Response | n/a   |                |            |
| See Also | [ ASX ], GOWHEN, INDUST, SMPER, TAS, TASF, TRGFN, TSTAT, VARB |                |            |

---

Use the AS operator to assign the axis status bits for a specific axis to a binary variable, or to make a comparison against a binary or hexadecimal value.

To make a comparison against a binary value, the letter b (b or B) must be placed in front of the value that the axis status is being compared against. The binary value itself must only contain ones, zeros, or Xs (1, 0, X, x). To make a comparison against a hexadecimal value, the letter h (h or H) must be placed in front of the value that the axis status is being compared against. The hexadecimal value itself must only contain the letters A through F, and the numbers 0 through 9. When using AS, an axis specifier must always proceed it, or else it will default to axis 1. Valid axis specifiers are 1, 2, 3, or 4 (1AS, 2AS, 3AS, or 4AS). The function of each axis status bit is shown below. An "x" identifies products to which the function is applicable.

---

| <b>Bit #</b><br>(left to right) | <b>Function</b> (1/0)   |
|---------------------------------|---|
| 1                               | Moving/Not Moving. This bit is set only when motion is <u>commanded</u> on the axis. The motor may still be "moving" (e.g., due to end-of-move settling).   |
| 2                               | Negative/positive-direction   |
| 3                               | Accelerating/Not Accelerating. This bit does not indicate deceleration (bit is set to 0 during decel); to check if the axis is decelerating, the state of AS bits 1, 3 and 4 should be: AS1x00.   |
| 4                               | At Velocity/Not at Velocity   |
| 5                               | Home Successful (HOM) (YES/NO)  |
| 6                               | Absolute/Incremental (MA1/MA0)  |
| 7                               | Continuous/Preset (MC1/MC0)   |
| 8                               | Jog Mode/Not Jog Mode (JOG)   |
| 9                               | Joystick Mode/Not Joystick Mode (JOY1/JOY0)   |
| 10                              | RESERVED  |
| 11                              | RESERVED  |
| 12                              | Stall Detected (YES/NO). This bit is not usable until Stall Detect is enabled with ESTALL1 command.   |
| 13                              | Drive Shut Down (YES/NO)  |
| 14                              | Drive Fault occurred (YES/NO). A drive fault cannot be detected (this bit is always 0) until the drive fault input check is enabled with DRFEN1. <u>Note:</u> ASX bit 4 reports the hardware state of the drive fault input, regardless of DRFEN or DRIVE.                    |
| 15                              | Positive-direction Hardware Limit Hit (YES/NO)  |
| 16                              | Negative-direction Hardware Limit Hit (YES/NO)  |
| 17                              | Positive-direction Software Limit Hit (YES/NO)  |
| 18                              | Negative-direction Software Limit Hit (YES/NO)  |
| 19                              | RESERVED  |
| 20                              | RESERVED  |
| 21                              | RESERVED  |
| 22                              | RESERVED  |
| 23                              | Position Error Exceeded (SMPER) (YES/NO). Servo axes only.  |
| 24                              | In Target Zone (defined with STRGTD & STRGTV) (YES/NO). Servo axes only. This bit is set only after the <i>successful completion</i> of a move (if STRGTD and STRGTV criteria have been satisfied). This bit is usable even if the Target Zone mode is not enabled (STRGTE0). |

---

| Bit #<br>(left to right) | Function (1/∅)  |
|--------------------------|---|
| 25                       | Target Zone Timeout occurred (STRGTT) (YES/NO). Servo axes only.  |
| 26                       | Change in motion is suspended pending GOWHEN (YES/NO). This bit is cleared if the GOWHEN condition is true, or if STOP (!S) or KILL (!K or ^K) is executed. |
| 27                       | RESERVED  |
| 28                       | Registration move initiated by trigger since last GO command. This bit is cleared with the next GO command.   |
| 29                       | RESERVED  |
| 30                       | Pre-emptive (OTF) GO or Registration profile not possible   |
| 31                       | RESERVED  |
| 32                       | RESERVED  |

**Syntax:** VARBn=aAS where n is the binary variable number and a is the axis identifier, or AS can be used in an expression such as IF(1AS=b1101), or IF(1AS=h7F). If it is desired to assign only one bit of the axis status value to a binary variable, instead of all 32, the bit select (.) operator can be used. The bit select, in conjunction with the bit number, is used to specify a specific axis status bit (e.g., VARB1=1AS.12 assigns axis 1 status bit 12 to binary variable 1).

**Example:**

```

VARB1=1AS           ; Axis status for axis 1 assigned to binary variable 1
VARB2=1AS.12       ; Axis 1 status bit 12 assigned to binary variable 2
VARB2               ; Response, if bit 12 is set to 1, is
                    ; "*VARB2=XXXX_XXXX_XXX1_XXXX_XXXX_XXXX_XXXX_XXXX"
IF(4AS=b111011X11) ; If the axis status for axis 4 contains 1's for
                    ; inputs 1,2,3,5,6,8,and 9, and a 0 for bit location 4,
                    ; do the IF statement
TREV               ; Transfer revision level
NIF               ; End if statement
IF(2AS=h7F00)     ; If the axis status for axis 2 contains 1's for
                    ; inputs 1,2,3,5,6,7,and 8, and 0's for every other bit
                    ; location, do the IF statement
TREV               ; Transfer revision level
NIF               ; End if statement

```

---

## [ ASX ] Extended Axis Status

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Assignment or Comparison                  | <b>Product</b> | <b>Rev</b> |
| Syntax   | See below                                 | 6K             | 5.0        |
| Units    | n/a                                       |                |            |
| Range    | n/a                                       |                |            |
| Default  | n/a                                       |                |            |
| Response | n/a                                       |                |            |
| See Also | EFAIL, TASX, TASXF, [AS], TAS, TASF, VARB |                |            |

---

The Extended Axis Status (ASX) command is used to assign the axis status bits for a specific axis to a binary variable, or to make a comparison against a binary or hexadecimal value.

To make a comparison against a binary value, the letter b (b or B) must be placed in front of the value that the axis status is being compared against. The binary value itself must only contain ones, zeros, or Xs (1, 0, X, x). To make a comparison against a hexadecimal value, the letter h (h or H) must be placed in front of the value that the axis status is being compared against. The hexadecimal value itself must only contain the letters A through F, and the numbers 0 through 9. An "x" identifies products to which the function is applicable.

---

| Bit Assignment  |  |
|-----------------|--|
| (left to right) | Function (1 = yes, 0 = no)                 |
| 1-3             | RESERVED                                   |
| 4 *             | Drive Fault Input Active                   |
| 5 **            | Encoder Failure                            |
| 6               | Z-channel state (1 = active, 0 = inactive) |
| 7-32            | RESERVED                                   |

---

\* Bit #4 indicates the current hardware state of the drive fault input, even in the factory default power-up state —the drive is disabled (see DRIVE command) and the drive fault input is disabled (see DRFEN command).

\*\* Bit #5 requires the Encoder Failure detection be enabled for the particular axis (see EFAIL command); this bit is cleared with the EFAIL0 command.

**Syntax:** VARBn=ASX where n is the binary variable number, or ASX can be used in an expression such as IF(ASX=b1100), or IF(ASX=h70). If it is desired to assign only one bit of the axis status value to a binary variable, instead of all 32, the bit select (.) operator can be used. The bit select, in conjunction with the bit number, is used to specify a specific axis status bit (e.g., VARB1=ASX.3 assigns axis 1 status bit 3 to binary variable 1).

**Example:**

```
VARB1=ASX           ; Extended Axis status for axis 1 assigned to
                    ; binary variable 1
VARB2=ASX.3        ; Extended Axis 1 status bit 3 assigned to
                    ; binary variable 2
VARB2              ; Response if bit 3 is set to 1:
                    ; "**VARB2=XX1X_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX"
IF(ASX=b101XXXXX) ; If the extended axis status for axis 1 contains 1's
                    ; for bits 1 and 3, and a 0 for bit location 2, do the
                    ; IF statement
TREV              ; Transfer revision level
NIF              ; End if statement
```

## [ ATAN() ] Arc Tangent

|          |   |
|----------|---|
| Type     | Operator (Trigonometric)                    |
| Syntax   | VARi=ATAN(r)                                |
| Units    | r = real number                             |
| Range    | 0.00000 to ±999,999,999                     |
| Default  | none  |
| Response | n/a   |
| See Also | [=], [COS], [PI], RADIAN, [SIN], [TAN], VAR |

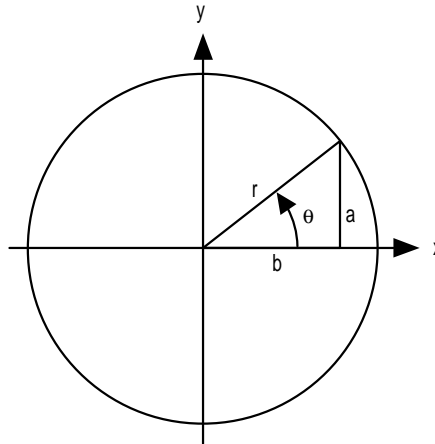
|                |            |
|----------------|------------|
| <b>Product</b> | <b>Rev</b> |
| 6K             | 5.0        |

This Arc Tangent (ATAN) operator is used to calculate the inverse tangent of a real number. If “a” and “b” are coordinates of a point on a circle of radius “r”, then the angle of measure “θ” can be defined by the equation:

$$\theta = \arctan \frac{a}{b}$$

The result of the ATAN command will either be in degrees or radians, depending on the RADIAN command.

To convert radians to degrees, use the formula:  
 $360^\circ = 2\pi$  radians.



$$\sin \theta = \frac{a}{r}$$

$$\cos \theta = \frac{b}{r}$$

$$\tan \theta = \frac{a}{b}$$

**Syntax:** VARi=ATAN(r) where i is the variable number and r is a real number value. Parentheses ( ) must be used with the ATAN command. **The result will be specified to 2 decimal places in either radians or degrees.**

**Example:**

```
RADIAN1           ; Enable radian mode
VAR1=ATAN(0.75)   ; Set variable 1 equal to the inverse tangent of 0.75 radians
```

## AXSDEF Axis Definition

|          |   |
|----------|---|
| Type     | Controller Configuration                        |
| Syntax   | <!><@>AXSDEF<b><b><b><b><b><b><b><b>            |
| Units    | n/a   |
| Range    | b = 0 (stepper), 1 (servo), or X (don't change) |
| Default  | 1 (servo)                                       |
| Response | AXSDEF: *11111111                               |
| See Also | DRIVE   |

|                |            |
|----------------|------------|
| <b>Product</b> | <b>Rev</b> |
| 6K             | 5.0        |

The Axis Definition (AXSDEF) command identifies the type of drive (servo or stepper) to which the controller axis is connected. The drive must be disabled (DRIVE0) for the AXSDEF command to function properly. Stepper drives receive their positioning information via step and direction signals. Servo drives receive their positioning commands via a ±10 volt signal. The AXSDEF setting is automatically saved in battery backed RAM.

The value of AXSDEF disables command fields that are not appropriate for that type of drive. For example, an axis configured as a stepper cannot be affected by a Servo Proportional Gain (SGP) command. The report back of non-applicable commands contains “-” in the field for that axis.

|                                  |       |
|----------------------------------|-------|
| AXSDEF0 — Stepper Only Commands: |       |
| DRES                             | FMAXA |
| ENCCNT                           | FMAXV |
| ESDB                             | PULSE |
| ESK                              |       |
| ESTALL                           |       |

|                                |        |        |        |
|--------------------------------|--------|--------|--------|
| AXSDEF1 — Servo Only Commands: |        |        |        |
| ANIFB                          | KDRIVE | PER    | SFB    |
| SGP                            | SGI    | SGV    | SGVF   |
| SGAF                           | SGILIM | SOFFS  | SMPER  |
| SGSET                          | SGENB  | STRGTD | STRGTE |
| STRGTT                         | STRGTV | TFB    | TGAIN  |
| TPER                           | TSGSET | TSTLT  |        |

**NOTE:** If you change the axis definition, be sure to verify or set all motion settings and scaling values to achieve the expected performance.

---

## BAUD

### Baud Rate

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Communication Interface                             | <b>Product</b> | <b>Rev</b> |
| Syntax   | BAUD<i>   | 6K             | 5.0        |
| Units    | i = Baud rate                                       |                |            |
| Range    | i = 1200, 2400, 4800, 9600, 19200, 38400, or 115200 |                |            |
| Default  | 9600  |                |            |
| Response | BAUD *BAUD9600                                      |                |            |
| See Also | ADDR, E, PORT                                       |                |            |

---

BAUD establishes the baud rate for the “RS-232” (COM1) or the “RS-232/485” (COM2) serial port, as selected by the last PORT command. The default is 9600 baud (BAUD9600). The BAUD setting is automatically saved in battery backed RAM. **NOTE:** Changing the baud rate for the currently used port will result in a loss of communication until the baud rate of the terminal is changed accordingly.

#### Example:

```
PORT2          ; Select COM2 ("RS-232/485") port
BAUD38400      ; Set the baud rate for COM2 to 38400 baud
```

---

## BOT

### Beginning of Transmission Characters

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Communication Interface                   | <b>Product</b> | <b>Rev</b> |
| Syntax   | <!>BOT<i>,<i>,<i>                         | 6K             | 5.0        |
| Units    | n/a                                       |                |            |
| Range    | i = 0 - 256                               |                |            |
| Default  | 0,0,0                                     |                |            |
| Response | BOT: *BOT0,0,0                            |                |            |
| See Also | EOT, ERROK, ERBAD, PORT, DRPCHK,EOL, ], [ |                |            |

---

The Beginning of Transmission Characters (BOT) command designates the characters to be placed at the beginning of every response. Up to 3 characters can be placed before the first line of a multi-line response, or before all single-line responses. The characters are designated with their ASCII equivalent. For example, a carriage return is ASCII 13, a line feed is ASCII 10, a Ctrl-Z is ASCII 26, and no terminating character is designated with a zero. Note that ASCII 256 means ∅∅ is transmitted.

For a more complete list of ASCII Equivalents, refer to the ASCII Table in Appendix B.

#### Example:

```
BOT13,10,26    ; Place a carriage return, line feed, and Ctrl-Z before
                ; the first line of a multi-line response, and before
                ; all single line responses
```

---

## BP

### Set a Program Break Point

|          |  |                |            |
|----------|--|----------------|------------|
| Type     | Program Flow Control or Program Debug Tool | <b>Product</b> | <b>Rev</b> |
| Syntax   | <!>BP<i>                                   | 6K             | 5.0        |
| Units    | i = break point number                     |                |            |
| Range    | 1 - 32                                     |                |            |
| Default  | n/a  |                |            |
| Response | n/a  |                |            |
| See Also | BREAK, C, HALT, K, S, [ SS ], TSS          |                |            |

---

The Break Point (BP) command allows the programmer to set a place in the program where command processing will halt and a message will be transmitted to the PC. There are 32 break points available, BP1 to BP32, all transmitting the message \*BREAKPOINT NUMBER x<cr> where x is the break point number.

After halting at a break point, command processing can be resumed by issuing a continue (!C) command.

The break point command is useful for stopping a program at specific locations in order to test status for debugging or other purposes.

**Example:**

```

DEF prog1          ; Begin definition of program named prog1
D50000,1000       ; Set distance to 50000 units on axis 1, and 1000 units on axis 2
MA1100           ; Absolute mode for axes 1 and 2
GO1100           ; Initiate motion on axes 1 and 2
IF(1PC>40000)    ; Compare axis 1 commanded position to 40000
BP1              ; If the motor position is > 40000 units, set break point #1
NIF              ; End IF statement
D80000,2000      ; Set distance to 80000 units on axis 1, and 2000 units on axis 2
GO1100           ; Initiate motion on axes 1 and 2
BP2              ; Set break point #2
END              ; End program definition
RUN prog1        ; Execute program prog1

```

If the IF statement evaluates true, the message \*BREAKPOINT NUMBER 1 will be transferred out. A !C command must be issued before processing will continue. Once processing has continued, the second break point command will be encountered, again the message \*BREAKPOINT NUMBER 2 will be transferred out, and processing of commands will pause until a second !C command is received.

---

## BREAK Terminate Program Execution

|          |                          |         |     |
|----------|--------------------------|---------|-----|
| Type     | Program Flow Control     | Product | Rev |
| Syntax   | <!>BREAK                 | 6K      | 5.0 |
| Units    | n/a                      |         |     |
| Range    | n/a                      |         |     |
| Default  | n/a                      |         |     |
| Response | n/a                      |         |     |
| See Also | BP, C, GOSUB, HALT, K, S |         |     |

---

The BREAK command terminates program execution when processed. This command allows the user to terminate a program based upon a condition, or at any other particular point in the program where it is necessary to end the program. If the program terminated was called from another program, control will be passed to the calling program. This command is useful when debugging a program.

To terminate all program processing, use the HALT command.

**Example:**

```

DEF prog1          ; Define a program called prog1
GO1000            ; Initiate motion on axis 1
GOSUB prog2       ; Gosub to subroutine named prog2
GO0100           ; Initiate motion on axis 2
END              ; End program definition
DEF prog2         ; Define a program called prog2
GO1110           ; Initiate motion on axes 1, 2, and 3
IF(IN=b1X0)      ; IF condition is: status of trigger input 1 is
                  ; active (1) and trigger input 3 is inactive (0)
BREAK            ; If condition is true break out of program
ELSE             ; Else part of if condition
TPE              ; If condition does not come true, transfer position of
                  ; all encoders
NIF              ; End If statement
END              ; End program definition
RUN prog1        ; Execute program prog1
;
; Upon completion of motion on axis 1, subroutine prog2 is called. If inputs 1
; and 3 are in the correct state when the subroutine is entered, the subroutine
; will be terminated and returned to prog1, where motion on axis 2 will be
; initiated.

```

---

## C Continue Command Execution

|          |                                  |         |     |
|----------|----------------------------------|---------|-----|
| Type     | Program Flow Control             | Product | Rev |
| Syntax   | !C                               | 6K      | 5.0 |
| Units    | n/a                              |         |     |
| Range    | n/a                              |         |     |
| Default  | n/a                              |         |     |
| Response | n/a                              |         |     |
| See Also | BP, COMEXR, COMEXS, INFNC, PS, S |         |     |

---

The Continue (!C) command ends a pause state (PS), a break point (BP) condition, or a stopped (S) condition. When the controller is in a paused state or at a break point, no commands from the command buffer are executed. All immediate commands, however, are still processed. By sending a !C command, command processing will resume, starting with the first command after the PS command or the BP command. If a stop (S) command has been issued, motion and command processing can be resumed by issuing a !C command, only if COMEXS has been enabled.

### Example:

```
PS           ; Stop execution of command buffer until !C command
MA0XXX      ; Incremental mode for axis 1
D10000      ; Set distance to 10000 units on axis 1
GO1000      ; Initiate motion on axis 1
D,20000     ; Set distance to 20000 units on axis 2
GO0100      ; Initiate motion on axis 2
```

No buffered commands after the PS command will be executed until a !C command is received.

```
!C           ; Restart execution of command buffer
DEF prog1   ; Begin definition of program named prog1
D50000,1000 ; Set distance to 50000 units on axis 1, & 1000 units on axis 2
MA00       ; Set axes 1 and 2 to the incremental mode
GO11       ; Initiate motion on axes 1 and 2
IF(VAR1>6) ; Compare VAR1>6
BP1        ; If the motor position is > 50000 units, set break point #1
NIF        ; End IF statement
GO11       ; Initiate motion on axes 1 and 2
BP2        ; Set break point #2
END        ; End program definition
RUN prog1   ; Execute program prog1
```

If the IF statement evaluates true, the message BREAKPOINT NUMBER 1 will be transferred out. A !C command must be issued before processing will continue. Once processing has continued, the second break point command will be encountered, again the message BREAKPOINT NUMBER 2 will be transferred out, and processing of commands will pause until a second !C command is received.

```
COMEXS1     ; Enable command processing on stop
D50000,1000 ; Set distance to 50000 units on axis 1, & 1000 units on axis 2
GO1100     ; Initiate motion on axes 1 and 2
!S         ; Stop motion on all axes
```

When the 6K Series product processes the !S command, motion on all axes will be stopped. If the desired distance has not been reached, motion can be resumed by issuing the !C command. If motion and command processing are to stop, a Kill (!K) command can be issued.

---

## CMDDIR      Commanded Direction Polarity

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Controller Configuration  | <b>Product</b> | <b>Rev</b> |
| Syntax   | <@><a>CMDDIR<b><b><b><b><b><b><b><b>  | 6K             | 5.0        |
| Units    | b = polarity bit  |                |            |
| Range    | 0 (normal polarity), 1 (reverse polarity) or X (don't change)   |                |            |
| Default  | 0   |                |            |
| Response | CMDDIR    *CMDDIR0000_0000<br>1CMDDIR   *1CMDDIR0   |                |            |
| See Also | [ AS ], DRIVE, ENCPOL, [ FB ], [ PANI ], [ PCE ], [ PE ],<br>[ PER ], PSET, SFB, TAS, TFB, TPANI, TPCE, TPE, TPER |                |            |

---

The CMDDIR command allows you to reverse the direction that the controller considers to be the “positive” direction; this also reverses the polarity of the counts from the feedback devices. Thus, using the CMDDIR command, you can reverse the referenced direction of motion without the need to (a) change the connections to the drive and the feedback device, or (b) change the sign of all the motion-related commands in your program.

|              |
|--------------|
| <b>NOTES</b> |
|--------------|

- **SERVO AXES:** Before changing the commanded direction polarity, make sure there is a direct correlation between the commanded direction and the direction of the feedback source counts (i.e., a positive commanded direction from the controller must result in positive counts from the feedback device). Refer to the ENCPOL command description for information on changing encoder polarity.
  - Once you change the commanded direction polarity, you should swap the end-of-travel limit connections to maintain a positive correlation with the commanded direction.
- 

The CMDDIR command is automatically saved in non-volatile memory.

**The CMDDIR command cannot be executed while motion is in progress or while the drive/valve is enabled.** For example, you could wait for motion to be complete (indicated when AS bit #1 is a zero) and then use the DRIVE command to disable the appropriate axis before executing the CMDDIR command.

---

## COMEXC      Continuous Command Processing Mode

|          |  |                |            |
|----------|--|----------------|------------|
| Type     | Command Buffer Control   | <b>Product</b> | <b>Rev</b> |
| Syntax   | <!>COMEXC<b>   | 6K             | 5.0        |
| Units    | b = 0, 1 or X  |                |            |
| Range    | 0 = Disable, 1 = Enable, X = don't change  |                |            |
| Default  | 0  |                |            |
| Response | COMEXC:   *COMEXC0   |                |            |
| See Also | [ ! ], A, AA, AD, ADA, COMEXL, COMEXS, D, ERRORP, FOLRD,<br>FOLRN, GO, GOWHEN, MA, MC, V |                |            |

---

This command enables (COMEXC1) or disables (COMEXC0) Continuous Command Execution Mode. Normally, when a motion command is received, command processing is temporarily paused until the motion is complete. In continuous command execution mode, however, command processing continues while motion is taking place. **NOTE:** Command processing will be slower and **some** motion parameters cannot be changed while motion is in progress; for a complete list of motion parameters that cannot be changed while motion is in progress, refer to the Restricted Commands During Motion section in Chapter 1 of the *Programmer's Guide*.

The Continuous Command Processing Mode is useful in the following situations:

- When trying to check the status of inputs while the 6K Series product is commanding motion.
- Performing calculations ahead of time, possibly decreasing cycle time.
- Executing buffered on-the-fly acceleration (A, AA), and deceleration (AD, ADA), distance (D), positioning mode (MA & MC), Following ratio (FOLRD & FOLRN), and velocity (V) changes. (The buffered A, AA, AD, ADA, D, FOLRD, FOLRN, MA, MC, or V change can be executed only with a buffered Go (GO) command.) For more information about on-the-fly motion changes, refer to the *Programmer's Guide*.
- Pre-processing the next move while the current move is in progress (see CAUTION note below). This reduces the processing time for the subsequent move to only a few microseconds.

**CAUTION: Avoid Executing Moves Prematurely**

With continuous command execution enabled (COMEXC1), if you wish motion to stop before executing the subsequent move, place a WAIT(AS.1=b0) statement before the subsequent GO command. If you wish to ensure the load settles adequately before the next move, use the WAIT(AS.24=b1) command instead (this requires you to define end-of-move settling criteria — see STRGTE command or *Programmer's Guide* for details).

**Example:**

```

VAR1=2000      ; Set variable 1 = 2000
VAR2=0         ; Set variable 2 = 0
COMEXC1       ; Enable continuous command execution mode
L50           ; Loop 50 times
D50000,(VAR1) ; Set distance to 50000 units for axis 1, VAR1 value for axis 2
GO1100       ; Initiate motion on axes 1 and 2
; Normally at this point, the 6K controller would wait for the motion on axes
; 1 & 2 to complete before processing the next command. However, with continuous
; command execution enabled (COMEXC1), processing will continue with the
; statements that follow.
REPEAT        ; Beginning of REPEAT..UNTIL() expression
IF(IN.1=b1)   ; Check for onboard input #1 (trigger A for axis 1)
              ; becoming active
VAR1=VAR1+10  ; If it does, increase variable 1 by 10
VAR2=1        ; Variable 2 is used as a flag
NIF           ; End IF statement
UNTIL(MOV=b0 OR VAR2=5) ; Exit REPEAT loop if variable 2 equals 5 or if
              ; motion is complete on axis 1
VAR2=0        ; Reset flag value, variable 2 = 0
LN           ; End loop
COMEXC0       ; Disable continuous command mode

```

**On-the-fly Velocity, Acceleration and Deceleration Change Example:**

```

DEF vsteps    ; Begin definition of program vsteps
COMEXC1       ; Enable continuous command execution mode
MC1           ; Set axis 1 mode to continuous
A10           ; Set axis 1 acceleration to 10 rev/sec/sec
V1           ; Set axis 1 velocity to 1 rps
GO1           ; Initiate axis 1 move (Go)
WAIT(1VEL=1)  ; Wait for motor to reach continuous velocity
T3           ; Time delay of 3 seconds
A50           ; Set axis 1 acceleration to 50 rev/sec/sec
V10          ; Set axis 1 velocity to 10 rps
GO1           ; Initiate axis 1 move (Go)
T5           ; Time delay of 5 seconds
S1           ; Initiate stop of axis 1 move
WAIT(MOV=b0)  ; Wait for motion to completely stop on axis 1
COMEXC0       ; Disable continuous command execution mode
END           ; End definition of program vsteps

```

---

## COMEXL Continue Execution on Limit

|          |  |                |            |
|----------|--|----------------|------------|
| Type     | Command Buffer Control                         | <b>Product</b> | <b>Rev</b> |
| Syntax   | <!><@><a>COMEXL<b><b><b><b><b><b><b><b>        | 6K             | 5.0        |
| Units    | b = 0, 1 or X                                  |                |            |
| Range    | 0 = Disable, 1 = Enable, X = don't change      |                |            |
| Default  | 0  |                |            |
| Response | COMEXL: *COMEXL0000_0000<br>1COMEXL: *1COMEXL0 |                |            |
| See Also | COMEXC, COMEXS, ERROR, LH, LHLVL, LS           |                |            |

---

This command determines whether the command buffer will be saved upon hitting a hardware end-of-travel limit (LH), or a soft limit (LS). If save command buffer on limit is enabled (COMEXL1), then all commands following the command currently being executed will remain in the command buffer when a limit is hit. If save command buffer on limit is disabled (COMEXL0), then every command in the buffer will be discarded, and program execution will be terminated.

### Example:

```
COMEXL0010      ; Save the command buffer only if the limit on axis 3 is hit.  
                ; Hitting a limit on any other axis will dump the command buffer.
```

---

## COMEXR Continue Motion on Pause/Continue Input

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Command Buffer Control                    | <b>Product</b> | <b>Rev</b> |
| Syntax   | <!>COMEXR<b>                              | 6K             | 5.0        |
| Units    | b = 0, 1 or X                             |                |            |
| Range    | 0 = disable, 1 = enable, X = don't change |                |            |
| Default  | 0   |                |            |
| Response | COMEXR: *COMEXR0                          |                |            |
| See Also | C, COMEXS, INFNC, LIMFNC                  |                |            |

---

The Continue Motion on Pause/Continue (COMEXR) command determines the functionality of programmable inputs defined as pause/continue inputs with the INFNCi-E or LIMFNCi-E command. In both cases, when the input is activated (exception: an axis-specific step input will not dump the buffer), the current command being processed will be allowed to finish executing.

COMEXR0: Upon receiving a pause input, only program execution is paused; any motion in progress will continue to its predetermined destination. Releasing the pause input or issuing a !C command will resume program execution.

COMEXR1: Upon receiving a pause input, both motion and program execution will be paused; the motion stop function is used to halt motion. *After motion stops*, you can release the pause input or issue a !C command to resume motion and program execution.

### Example:

```
COMEXR1      ; Allow both motion and program execution to be paused upon  
             ; receiving a pause input  
2INFNC1-E    ; Define input 1 on I/O brick 2 as a pause/continue input
```

---

## COMEXS Continue Execution on Stop

|          |  |         |     |
|----------|--|---------|-----|
| Type     | Command Buffer Control                   | Product | Rev |
| Syntax   | <!>COMEXS<i>                             | 6K      | 5.0 |
| Units    | i = function identifier                  |         |     |
| Range    | 0, 1, or 2                               |         |     |
| Default  | 0  |         |     |
| Response | COMEXS: *COMEXS0                         |         |     |
| See Also | COMEXC, COMEXL, COMEXR, INFNC, LIMFNC, S |         |     |

---

The Continue Execution on Stop (COMEXS) command determines whether the command buffer will be saved upon receiving a Stop command (!S or !S1111) or an external stop input (INFNCi-D or LIMFNCi-D).

COMEXS0: Upon receiving a stop input or Stop command, motion will decelerate at the preset AD/ADA value, every command in the buffer will be discarded (exception: an axis-specific stop input will not dump the buffer), and program execution will be terminated.

COMEXS1: Upon receiving a stop input or Stop (!S or !S1111) command, motion will decelerate at the preset AD/ADA value, command execution will be paused, and all commands following the command currently being executed will remain in the command buffer.

Resuming program execution (*only after motion is stopped*):

- Whether stopping as a result of a stop input or Stop (!S or !S1111) command, you can resume program execution by issuing an immediate Continue (!C) command or by activating a pause/resume input (a programmable input configured with the INFNCi-E or LIMFNCi-E command).
- If you are resuming after a stop input or !S1111 command, the move in progress will **not** be saved.
- If you are resuming after a !S command, you will resume the move in progress at the point in which the !S command was received by the processor.

COMEXS2: Upon receiving a stop input or Stop command, motion will decelerate at the preset AD/ADA value, every command in the buffer will be discarded, and program execution will be terminated, but the INSELP value is retained. This allows external program selection, via inputs defined with the INFNCi-B (or LIMFNCi-B) or INFNCi-iP (or LIMFNCi-iP) commands, to continue.

### Example:

COMEXS1 ; Save the command buffer upon a stop input or stop command

## [ COS() ] Cosine

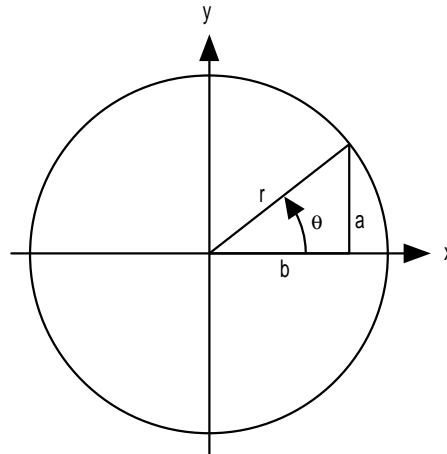
|          |  |                |            |
|----------|--|----------------|------------|
| Type     | Operator (Trigonometric)                             | <b>Product</b> | <b>Rev</b> |
| Syntax   | COS(r) (see below)                                   | 6K             | 5.0        |
| Units    | r = radians or degrees (depending on RADIAN command) |                |            |
| Range    | r = 0.00000 - ±17500                                 |                |            |
| Default  | n/a  |                |            |
| Response | n/a  |                |            |
| See Also | [ ATAN ], [ PI ], RADIAN, [ SIN ], [ TAN ], VAR      |                |            |

Use this operator to calculate the cosine of a number given in radians or degrees (see RADIAN command). If “a” and “b” are coordinates of a point on a circle of radius “r”, then the angle of measure “θ” can be defined by the equation:

$$\cos \theta = \frac{b}{r} \quad (\text{see illustration at right})$$

If a value is given in radians and a conversion is needed to degrees, or vice-versa, use the formula:

$$360^\circ = 2\pi \text{ radians.}$$

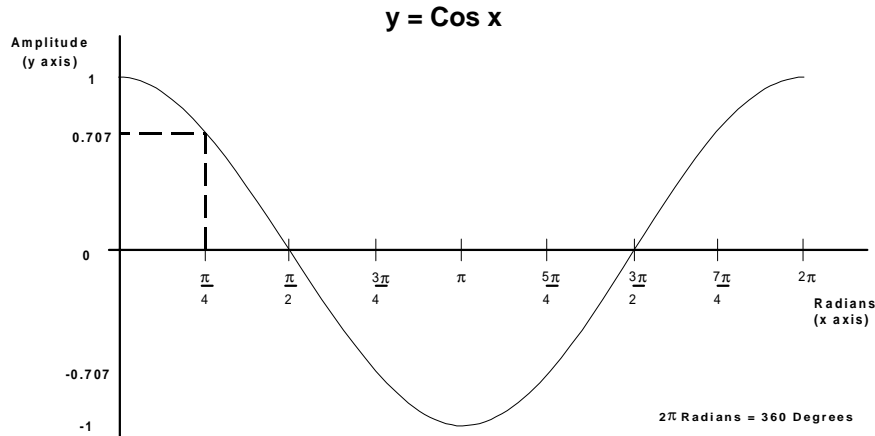


$$\sin \theta = \frac{a}{r}$$

$$\cos \theta = \frac{b}{r}$$

$$\tan \theta = \frac{a}{b}$$

The graph to the right shows the amplitude of **y** on the unit circle for different values of **x**.



**Syntax:** VARi=COS(r) where i is the variable number and r is a value in either radians or degrees depending on the RADIAN command. Parentheses ( ) must be placed around the COS operand.  
**The result will be specified to 5 decimal places.**

**Example:**  
 VAR1=5 \* COS(PI/4) ; Set variable 1 equal to 5 times the cosine of  
 ; π divided by 4