

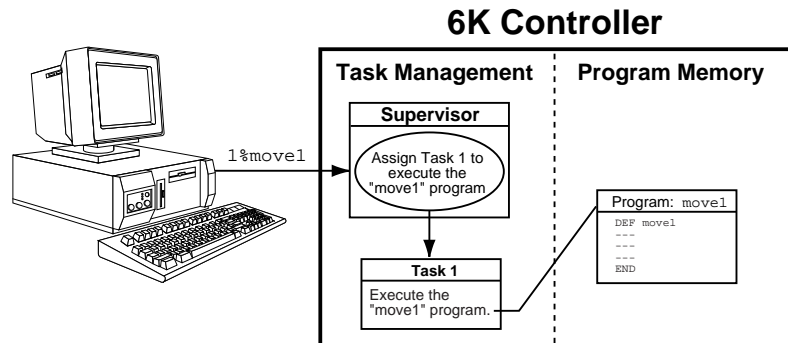
## % Task Identifier

|          |                 |
|----------|-----------------|
| Type     | Multi-Tasking   |
| Syntax   | i%<command>     |
| Units    | i = task number |
| Range    | 1-10            |
| Default  | 1               |
| Response | n/a             |

**Product Rev**  
6K 5.0

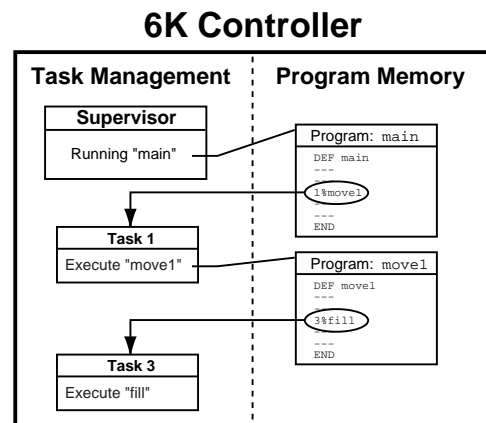
See Also LOCK, [SWAP], [TASK], TSKAX, TSKTRN, TSWAP, TTASK

Use the Task Identifier (%) prefix to specify that the associated command will **affect** the indicated task number. For most simple multi-tasking applications, the % prefix is used to start a program running in a specific task. For example, the drawing on the right illustrates how the 1%move1 command starts the program called "move1" in task 1 (specified with the 1% prefix).



Because the % prefix specifies the task number that the associated command will affect, new tasks can be started from within other tasks, as shown in the drawing on the right. →

Within a program in a task, it is not necessary to use the % prefix unless trying to initiate a program or command in a different task. For example, if the fill program running in task 3 executes a COMEXC1 command, only task 3 is placed into COMEXC1 mode. If the fill program running in task 3 also executes a 2%PS command, task 3 executes the command, but the program being executed in task 2 is paused, not task 3.



**How the Task Supervisor Works:** The "Task Supervisor" (also referred to as Task  $\emptyset$ ) is the main program execution environment. It contains the command buffer and parser. Immediate commands and commands executed from the communications buffer are implicitly directed to affect the supervisor unless explicitly directed to a task with the % prefix. Only the supervisor executes buffered commands from the communications buffer. If the supervisor is executing a program, incoming commands will be buffered, not executed. If the supervisor is not executing a program, it will execute commands from the input command buffer, even if the other tasks are executing programs. If a command in the command buffer has a task prefix, it is still executed by the supervisor, but affects the task specified by the prefix.

---

**[ ! ]****Immediate Command Identifier**

|          |                  |                |            |
|----------|------------------|----------------|------------|
| Type     | Operator (Other) | <b>Product</b> | <b>Rev</b> |
| Syntax   | !<command>       | 6K             | 5.0        |
| Units    | n/a              |                |            |
| Range    | n/a              |                |            |
| Default  | n/a              |                |            |
| Response | n/a              |                |            |
| See Also | COMEXC           |                |            |

---

The Immediate Command Identifier (!) changes a buffered command into an immediate command. All immediate commands are processed immediately, even before previously entered buffered commands.

All 6K Series commands are buffered.

The commands that use the ! identifier are identified in the **Syntax** portion of the command description.

**NOTE**

A command with the ! prefix cannot be stored in a program.

---

**[ @ ]****Global Command Identifier**

|          |                    |                |            |
|----------|--------------------|----------------|------------|
| Type     | Operator (Other)   | <b>Product</b> | <b>Rev</b> |
| Syntax   | @<command><field1> | 6K             | 5.0        |
| Units    | n/a                |                |            |
| Range    | n/a                |                |            |
| Default  | n/a                |                |            |
| Response | n/a                |                |            |
| See Also | INDAX              |                |            |

---

The Global Command Identifier (@) is used to set the value of all fields to the value entered only in the first field. For example, @A1 assigns the value 1 to all axes. All commands with multiple fields are able to use the Global Command Identifier. If you have any doubts about which commands can use the @ symbol, refer to the **Syntax** portion of the command description.

---

**;****Begin Comment**

|          |                      |                |            |
|----------|----------------------|----------------|------------|
| Type     | Operator (Other)     | <b>Product</b> | <b>Rev</b> |
| Syntax   | ;<this is a comment> | 6K             | 5.0        |
| Units    | n/a                  |                |            |
| Range    | n/a                  |                |            |
| Default  | n/a                  |                |            |
| Response | n/a                  |                |            |
| See Also | None                 |                |            |

---

The Begin Comment (;) command is used to comment application programs. The comment begins with a semicolon (;) and is terminated by a command delimiter. The comment is not stored in a program. An example of using the comment delimiter is as follows:

```
DEF pick ; Begin definition of program pick<cr>
```

| <b>\$</b> |   | <b>Label Declaration</b> | <b>Product</b> | <b>Rev</b> |
|-----------|---|--------------------------|----------------|------------|
| Type      | Operator (Other)                              |                          | 6K             | 5.0        |
| Syntax    | <!>\$<t>                                      |                          |                |            |
| Units     | t = text name                                 |                          |                |            |
| Range     | Text name of 6 characters or less             |                          |                |            |
| Default   | n/a   |                          |                |            |
| Response  | n/a   |                          |                |            |
| See Also  | DEF, DEL, END, GOSUB, GOTO, JUMP, RUN, TLABEL |                          |                |            |

The Label Declaration (\$) command defines the current location as the label specified. A label consists of 6 or fewer alpha-numeric characters and must start with an alpha-character, not a number. Labels can only be defined within a program or subroutine. The GOTO, GOSUB or JUMP commands can be used to branch to a label. The RUN command can also be used to start executing statements at a label. The label cannot be deleted by a DEL command. However, when the program that contains the label is deleted, all labels contained within the program will be deleted.

**NOTE:** The maximum number of labels possible is 600.

A label declaration cannot consist of any of the following characters:

!, \_, #, \$, %, ^, &, \*, (, ), +, -, {, }, \, |, ", :, ;, ', <, >, ,, ., ?, /, =

**NOTE:** A label cannot have the same name as a 6K Series command. For example, \$A and \$A123 are illegal labels.

**Example**

```

DEF pick          ; Begin definition of program called pick
GO1100           ; Initiate motion on axes 1 and 2
IF(VAR1=5)       ; If variable 1 = 5 then do commands between IF and ELSE,
                 ; otherwise commands between ELSE and NIF
GOTO pick1       ; Goto label pick1
ELSE             ; Else part of IF command
GOTO pick2       ; Goto label pick2
NIF              ; End IF command
$pick1           ; Label declaration for pick1
GO0011           ; Initiate motion on axes 3 and 4
BREAK           ; Break out of current subroutine or program
$pick2           ; Label declaration for pick2
GO1001           ; Initiate motion on axes 1 and 4
END              ; End program definition
RUN pick         ; Execute program named pick

```

---

## [ # ] Step Through a Program

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Operator (Other)                                  | <b>Product</b> | <b>Rev</b> |
| Syntax   | !#<i>   | 6K             | 5.0        |
| Units    | i = number of commands to execute from the buffer |                |            |
| Range    | i = 1 - 200                                       |                |            |
| Default  | 1   |                |            |
| Response | n/a   |                |            |
| See Also | DEF, HELP, STEP, TRACE, TRANS                     |                |            |

---

This command controls the execution of a program or sequence when the single step mode is enabled (STEP1). Each time you enter the !#<i> command followed by a delimiter, i commands in the sequence buffer will be executed. A !# followed by a delimiter will cause one command to be executed.

Single step mode can be advantageous when trying to debug a program.

### Example:

```
DEF tst           ; Begin definition of program named tst
@V1              ; Set velocity to 1 unit/sec on all axes
@A10             ; Set acceleration to 10 units/sec/sec on all axes
D1,2,3,4         ; Set distance to 1 unit on axis 1, 2 units on axis 2,
                ; 3 units on axis 3, and 4 units on axis 4
GO1101          ; Initiate motion on axes 1, 2, and 4
OUT11X1         ; Turn on on-board programmable outputs 1, 2, and 4,
                ; leave 3 unchanged
END              ; End program definition
STEP1           ; Enable single step mode
RUN tst         ; Execute program named tst
```

**NOTE:** After entering the command RUN no action will occur because single step mode has been enabled. Single step operation is as follows:

```
!#2             ; First 2 commands in the program tst are executed,
                ; commands to be executed are @V1 and @A10.
!#              ; Execute 1 command from program; command to execute is D1,2,3,4
!#1            ; Execute 1 command from program; command to be executed is GO1101
!#2            ; Execute 2 commands from program; commands to be executed are
                ; OUT11X1 and END
```

---

## ' Enter Interactive Data

|          |                                   |                |            |
|----------|-----------------------------------|----------------|------------|
| Type     | Operator (Other)                  | <b>Product</b> | <b>Rev</b> |
| Syntax   | !'<numeric data>                  | 6K             | 5.0        |
| Units    | Numeric data is command-dependent |                |            |
| Range    | Numeric data is command-dependent |                |            |
| Default  | n/a                               |                |            |
| Response | n/a                               |                |            |
| See Also | [ READ ], VARI, VARS              |                |            |

---

To enter data interactively, two operations must occur. First, numeric information must be requested. Requesting the numeric information is accomplished with the VAR<sub>x</sub>=READ<sub>y</sub> command. The <sub>x</sub> specifies the numeric variable to place the data into, and the <sub>y</sub> specifies the string variable to transmit before the data is entered. Numeric information can also be requested by placing the READ command in place of a command argument (e.g., A(READ1), 12.52, (READ2), 5.62). After the data has been requested, a numeric response must be provided. The numeric response must be preceded by the interactive data specifier (!') and followed by a delimiter (<cr> or <lf>).

Command processing will pause while waiting for data.

### Example:

```
VARS1="Enter the count > " ; Set string variable 1 equal to the message
VAR5=READ1                 ; Transmit string variable 1, and wait for numeric data in the
                            ; form of !'<data>. Once numeric data has been received, place
                            ; it in numeric variable 5
!'65.12                    ; Variable 5 will receive the value 65.12
```

---

**[ . ]****Bit Select**

|          |  |                |            |
|----------|--|----------------|------------|
| Type     | Operator (Other)   | <b>Product</b> | <b>Rev</b> |
| Syntax   | <command>.i  | 6K             | 5.0        |
| Units    | i = bit number   |                |            |
| Range    | Command-dependent  |                |            |
| Default  | None   |                |            |
| Response | n/a  |                |            |
| See Also | [ AS ], [ ER ], ERROR, [ IN ], INEN, INLVL, [ INO ], INTHW, LHLVL, [ LIM ], [ MOV ], ONIN, ONUS, OUT, OUTEN, OUTLVL, POUT, [ SS ], TAS, TER, TIN, TINO, TINT, TLIM, TOUT, TSS, TUS, [ US ] |                |            |

---

The Bit Select (.) operator specifies which bit to select. The primary purpose of this command is to let the user specify a specific bit (or range), instead of having to type in an entire bit string.

When using the bit operator in a comparison, the bit operator must always come to the left of the comparison. For example, the command IF(1AS.12=b1) is legal, but IF(b1=1AS.12) is illegal.

**Command Shortcut Examples** (affect only one binary bit location):

- Activate outputs at I/O location Brick 3, I/O point 9: 3OUT.9=1
- Enable analog input at I/O location Brick 2, I/O point 2: 2ANIEN.2=E
- Enable error-checking bit 6 for task 3: 3%ERROR.6=1

**Example:**

```
VARB2=ER.12      ; Error status bit 12 assigned to binary variable 2
VARB2            ; Response (if bit 12 is set to 1):
                 ; *VARB2=XXXX_XXXX_XXX1_XXXX_XXXX_XXXX_XXXX_XXXX
2OUT.5=1        ; Activate the output at location Brick 2, I/O point 5
```

---

**[ " ]****Begin and End String**

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Operator (Other)                          | <b>Product</b> | <b>Rev</b> |
| Syntax   | "<message>" (see below for possibilities) | 6K             | 5.0        |
| Units    | n/a                                       |                |            |
| Range    | n/a                                       |                |            |
| Default  | n/a                                       |                |            |
| Response | n/a                                       |                |            |
| See Also | DWRITE, VARS, WRITE, WRVARS               |                |            |

---

There are three commands that deal with string variables, or messages. The first of these commands is the VARS command. This command sets a string variable equal to a specific message (e.g., VARS1="Enter part count"). The message must be placed in quotes for it to be recognized. The same can be said for the WRITE and DWRITE commands. Their messages must also be placed in quotes (e.g., WRITE"Today is the first day of the rest of your life").

Syntax possibilities: VARSn="<message>" where n equals the string variable number  
WRITE"<message>"  
DWRITE"<message>"

There are three ASCII characters that cannot be used within the quotes (:, ", and ;). These characters can be specified in the string by using the backslash character (\) in combination with the ASCII decimal value for the character. For example, if you wanted to display the message "WHY ASK WHY" in quotes, you would use the following syntax: WRITE"\34WHY ASK WHY\34".

An ASCII table is provided in Appendix B. Common characters and their ASCII equivalent value:

| Character | Description     | ASCII Decimal Value             |
|-----------|-----------------|---------------------------------|
| <lf>      | Line Feed       | 10                              |
| <cr>      | Carriage Return | 13                              |
| "         | Quote           | 34                              |
| :         | Colon           | 58                              |
| ;         | Semi-colon      | 59                              |
| \         | Backslash       | 92 (cannot be used with DWRITE) |

---

## [ \ ] ASCII Character Designator

|          |                   |                |            |
|----------|-------------------|----------------|------------|
| Type     | Operator (Other)  | <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 | VAR, WRITE, WRVAR |                |            |

---

The ASCII Character Designator (\) operator is used to place a character in a string that is normally not represented by a keyboard character. The (\) operator can be used within the VAR or the WRITE commands. The syntax for the (\) operator is as follows:

WRITE "\<i>", Where <i> is the ASCII decimal equivalent of the character to be placed in the string.

VAR1="\<i>", Where <i> is the ASCII decimal equivalent of the character to be placed in the string.

There are three ASCII characters that cannot be used within the quotes (:, ;, and "). These characters must be specified in the string by using the backslash character (\) in combination with the ASCII decimal value for the character.

An ASCII table is provided in Appendix B. Common characters and their ASCII equivalent value:

| Character | Description     | ASCII Decimal Value |
|-----------|-----------------|---------------------|
| <lf>      | Line Feed       | 10                  |
| <cr>      | Carriage Return | 13                  |
| "         | Quote           | 34                  |
| :         | Colon           | 58                  |
| ;         | Semi-colon      | 59                  |
| \         | Backslash       | 92                  |

---

**Example:**

```
WRITE "cd\92AT6400\13\10" ;Displays: cd\AT6400<cr><lf>
```

---

## [ = ] Assignment or Equivalence

|          |  |                |            |
|----------|--|----------------|------------|
| Type     | Operator (Mathematical or Relational)  | <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 | [ > ], [ >= ], [ < ], [ <= ], [ <> ], [ AND ], IF, [ OR ], UNTIL, VAR, VARB, VARI, VARS, WAIT, WHILE |                |            |

---

The assignment or equivalence operator (=) is used to either assign a value to a variable, or compare two values and/or variables. The (=) operator is limited to 1 assignment operation per line. It is acceptable to state VAR1=25, but it is unacceptable to state VAR1=25=VAR2.

More than 1 equivalence operator can be used in a command; however, the total number of relational operators used in a line is limited by the command length limitation (80 characters), not the number of relational operators (e.g., the command IF (VAR1=1 AND VAR2=4 AND VAR3=4) is a legal command).

When (=) is used as an assignment operator, it can be used with these commands: VAR, VARI, VARB, VARS.

When (=) is used as an equivalence operator, it can be used with these commands: IF, WHILE, UNTIL, WAIT.

---

**[ > ]****Greater Than**

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Operator (Relational)   | <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 | [ = ], [ >= ], [ < ], [ <= ], [ <> ], [ AND ], IF, [ OR ], UNTIL, WAIT, WHILE |                |            |

---

The greater than (>) operator is used to compare two values. If the value on the left of the operator is greater than the value on the right of the operator, then the expression is *TRUE*. If the value on the left is less than or equal to the value on the right of the operator, then the expression is *FALSE*. The greater than operator (>) can only be used to compare two values.

More than one (>) operator can be used within a single command; however, the total command length is limited to 80 characters.

The (>) operator can be used in conjunction with the IF, WHILE, UNTIL, and WAIT commands.

Examples of valid commands are IF (VAR1>1) and WHILE (VAR1>1 AND VAR2>3). An example of an invalid command is IF (5>VAR1>1).

---

**[ >= ]****Greater Than or Equal**

|          |  |                |            |
|----------|--|----------------|------------|
| Type     | Operator (Relational)  | <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 | [ = ], [ > ], [ < ], [ <= ], [ <> ], [ AND ], IF, [ OR ], UNTIL, WAIT, WHILE |                |            |

---

The greater than or equal (>=) operator is used to compare two values. If the value on the left of the operator is greater than or equal to the value on the right of the operator, then the expression is *TRUE*. If the value on the left is less than the value on the right of the operator, then the expression is *FALSE*. The greater than or equal operator (>=) can only be used to compare two values.

More than one (>=) operator can be used within a single command; however, the total command length is limited to 80 characters.

The (>=) operator can be used in conjunction with the IF, WHILE, UNTIL, and WAIT commands.

Examples of valid commands are IF (VAR1>=1) and WHILE (VAR1>=1 AND VAR2>=3). An example of an invalid command is IF (5>VAR1>=1).

---

**[ < ]****Less Than**

|          |   |                |            |
|----------|---|----------------|------------|
| Type     | Operator (Relational)   | <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 | [ = ], [ > ], [ >= ], [ <= ], [ <> ], [ AND ], IF, [ OR ], UNTIL, WAIT, WHILE |                |            |

---

The less than (<) operator is used to compare two values. If the value on the left of the operator is less than the value on the right of the operator, then the expression is *TRUE*. If the value on the left is greater than or equal to the value on the right of the operator, then the expression is *FALSE*. The less than operator (<) can only be used to compare two values.

More than one (<) operator can be used within a single command; however, the total command length is limited to 80 characters.

The (<) operator can be used in conjunction with the IF, WHILE, UNTIL, and WAIT commands.

Examples of valid commands are IF (VAR1<1) and WHILE (VAR1<1 AND VAR2<3). An example of an invalid command is IF (1<VAR1<54).

---

| <b>[ &lt;= ]</b> |  | <b>Less Than or Equal</b> |            |
|------------------|--|---------------------------|------------|
| Type             | Operator (Relational)  | <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         | [ = ], [ > ], [ < ], [ >= ], [ <> ], [ AND ], IF, [ OR ], UNTIL, WAIT, WHILE |                           |            |

---

The less than or equal (<=) operator is used to compare two values. If the value on the left of the operator is less than or equal to the value on the right of the operator, then the expression is *TRUE*. If the value on the left is greater than the value on the right of the operator, then the expression is *FALSE*. The less than or equal operator (<=) can only be used to compare two values.

More than one (<=) operator can be used within a single command; however, the total command length is limited to 80 characters.

The (<=) operator can be used in conjunction with the IF, WHILE, UNTIL, and WAIT commands.

Examples of valid commands are IF (VAR1<=1) and WHILE (VAR1<=1 AND VAR2<=3). An example of an invalid command is IF (1<VAR1<=54).

---

| <b>[ &lt;&gt; ]</b> |   | <b>Not Equal</b> |            |
|---------------------|---|------------------|------------|
| Type                | Operator (Relational)   | <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            | [ = ], [ >= ], [ < ], [ <= ], [ AND ], IF, [ OR ], UNTIL, WAIT, WHILE |                  |            |

---

The not equal (<>) operator is used to compare two values. If the value on the left of the operator is not equal to the value on the right of the operator, then the expression is *TRUE*. If the value on the left is equal to the value on the right of the operator, then the expression is *FALSE*. The not equal operator (<>) can only be used to compare two values.

More than one (<>) operator can be used within a single command; however, the total command length is limited to 80 characters.

The (<>) operator can be used in conjunction with the IF, WHILE, UNTIL, and WAIT commands.

Examples of valid commands are IF (VAR1<>1) and WHILE (VAR1<>1 AND VAR2<=3). An example of an invalid command is IF (1<VAR1<>54).

---

|                |   |                |            |
|----------------|---|----------------|------------|
| <b>[ ( ) ]</b> | <b>Operation Priority Level</b>                 |                |            |
| Type           | Operator (Mathematical)                         | <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       | [ = ], [ - ], [ * ], [ / ], [ SQRT ], VAR, VARI |                |            |

---

The Operation Priority Level operators determines which operation to do first in a mathematical expression. For example, if you want to add 5 to 6 times 3, you can specify `VAR1=6*3+5` or `VAR1=5 + (6*3)`.

More than one set of parentheses can be used in a mathematical expression; however, they cannot be nested (e.g. `VAR1=(VAR2 * 3) * (3 + VAR4)` ).

---

|              |  |                |            |
|--------------|--|----------------|------------|
| <b>[ + ]</b> | <b>Addition</b>  |                |            |
| Type         | Operator (Mathematical)  | <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     | [ = ], [ ( ) ], [ - ], [ * ], [ / ], [ SQRT ], VAR, VARI, VARB |                |            |

---

The addition (+) operator adds the value to the left of the operator with the value to the right of the operator. The addition operator can only be used in conjunction with the VAR, VARI and VARB commands.

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level ( ( ) ) operators can be used; however, they cannot be nested.

Examples of valid commands: `VAR1=1+2+3+4+5+6+7+8+9`  
`VAR2=VAR1+1+(5*3)`  
`VARB1=b1101 + b11001`

---

|              |  |                |            |
|--------------|--|----------------|------------|
| <b>[ - ]</b> | <b>Subtraction</b>   |                |            |
| Type         | Operator (Mathematical)  | <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     | [ = ], [ ( ) ], [ + ], [ * ], [ / ], [ SQRT ], VAR, VARI, VARB |                |            |

---

The subtraction (-) operator subtracts the value to the right of the operator from the value to the left of the operator. The subtraction operator can only be used in conjunction with the VAR, VARI and VARB commands.

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level ( ( ) ) operators can be used; however, they cannot be nested.

Examples of valid command s: `VAR1=1-2-3-4-5-6-7-8-9`  
`VAR2=VAR1-1+(5*3)`  
`VARB1=b111101 - b11001`

| <b>[ * ]</b> |  | <b>Multiplication</b> |            |
|--------------|--|-----------------------|------------|
| Type         | Operator (Mathematical)  | <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     | [ = ], [ ( ) ], [ + ], [ - ], [ / ], [ SQRT ], VAR, VARI, VARB |                       |            |

The multiplication (\*) operator multiplies the value to the right of the operator with the value to the left of the operator. The multiplication operator can only be used in conjunction with the VAR, VARI and VARB commands (VARI integer values are truncated).

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level ( ) operators can be used; however, they cannot be nested.

Examples of valid commands: VAR1=1\*2\*3\*4\*5\*6\*7\*8\*9  
VAR2=VAR1-1+(5\*3)  
VARB1=b1111101 \* b11001

| <b>[ / ]</b> |  | <b>Division</b> |            |
|--------------|--|-----------------|------------|
| Type         | Operator (Mathematical)  | <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     | [ = ], [ ( ) ], [ + ], [ - ], [ * ], [ SQRT ], VAR, VARI, VARB |                 |            |

The division (/) operator divides the value to the left of the operator by the value on the right of the operator. The result of the division is specified to five decimal places (VARI integer variables are truncated). The division operator can only be used in conjunction with the VAR and VARB commands.

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level ( ) operators can be used; however, they cannot be nested.

Examples of valid commands: VAR1=1/2/3/4/5/6/7/8/9  
VAR2=VAR1-1/(5\*3)  
VARB1=b1111101 / b11001                      **DIVISION BY ZERO IS NOT ALLOWED.**

| <b>[ &amp; ]</b> |   | <b>Boolean And</b> |            |
|------------------|---|--------------------|------------|
| Type             | Operator (Bitwise)  | <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         | [ = ], [   ], [ ~ ], [ ^ ], [ << ], [ >> ], VAR, VARI, VARB |                    |            |

The Boolean And (&) operator performs a logical AND on the two values to the left and right of the operator when used with the VAR or VARI command. The Boolean And (&) performs a bitwise AND on the two values to the left and right of the operator when used with the VARB command.

For a logical AND (using VAR or VARI), the possible combinations are as follows:

|   |   |   |
|---|---|---|
| positive number & positive number                 | = | 1 |
| positive number & zero or a negative number       | = | 0 |
| zero or negative number & positive number         | = | 0 |
| zero or negative number & zero or negative number | = | 0 |
| Example: VAR1=5 & -1                              |   |   |
| Result: VAR1=0                                    |   |   |

For a bitwise AND (using VARB), the value on the left side of the & operator has each of its bits ANDed with the corresponding bit of the value on the right side of the operator. Each bit comparison will be composed of 9 possible combinations:

```

1 & 1 = 1          1 & X = X
1 & 0 = 0          X & 1 = X
0 & 1 = 0          0 & X = 0
0 & 0 = 0          X & 0 = 0
X & X = X

```

Example: VARB1=b0000 1000 & b1000 1011 1

Response to VARB1 is \*VARB1=0000\_1000\_XXXX\_XXXX\_XXXX\_XXXX\_XXXX\_XXXX

Example: VARB1=h32FD & h23

Response to VARB1 is \*VARB1=0100\_0100\_0000\_0000\_0000\_0000\_0000\_0000

Example: VARB1=h23 & b1101

Response to VARB1 is \*VARB1=0100\_XX00\_0000\_0000\_0000\_0000\_0000\_0000

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level ( ) operators can be used; however, they cannot be nested.

---

|              |   |                |            |
|--------------|---|----------------|------------|
| <b>[   ]</b> | <b>Boolean Inclusive Or</b>                                 | <b>Product</b> | <b>Rev</b> |
| Type         | Operator (Bitwise)  | 6K             | 5.0        |
| Syntax       | See Below   |                |            |
| Units        | n/a   |                |            |
| Range        | n/a   |                |            |
| Default      | n/a   |                |            |
| Response     | n/a   |                |            |
| See Also     | [ = ], [ & ], [ ~ ], [ ^ ], [ << ], [ >> ], VAR, VARI, VARB |                |            |

---

The Boolean Inclusive Or (|) operator performs a logical OR on the two values to the left and right of the operator when used with the VAR or VARI command. The Boolean Inclusive Or (|) performs a bitwise OR on the two values to the left and right of the operator when used with the VARB command.

For a logical OR (using VAR or VARI), the possible combinations are as follows:

```

positive number | positive number          = 1
positive number | zero or a negative number = 1
zero or negative number | positive number   = 1
zero or negative number | zero or negative number = 0

```

Example: VAR1=5 | -1

Result: VAR1=1

For a bitwise OR (using VARB), the value on the left side of the | operator has each of its bits ORed with the corresponding bit of the value on the right side of the operator. Each bit comparison will be composed of 9 possible combinations:

```

1 | 1 = 1          1 | X = 1
1 | 0 = 1          X | 1 = 1
0 | 1 = 1          0 | X = X
0 | 0 = 0          X | 0 = X
X | X = X

```

Example: VARB1=b1001 01X1 XX11 | b1000 1011 10

Response to VARB1 is \*VARB1=1001\_1111\_1X11\_XXXX\_XXXX\_XXXX\_XXXX\_XXXX

Example: VARB1=h1234 | hFAD31

Response to VARB1 is \*VARB1=1111\_0101\_1111\_1110\_1000\_0000\_0000\_0000

Example: VARB1=h23 | b1101 001X 001X 1X11

Response to VARB1 is \*VARB1=1101\_111X\_001X\_1X11\_XXXX\_XXXX\_XXXX\_XXXX

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level ( ) operators can be used; however, they cannot be nested.

| <b>[ ^ ]</b> |   | <b>Boolean Exclusive Or</b> |            |
|--------------|---|-----------------------------|------------|
| Type         | Operator (Bitwise)  | <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     | [ = ], [ & ], [ ~ ], [   ], [ << ], [ >> ], VAR, VARI, VARB |                             |            |

The Boolean Exclusive Or (^) operator performs a logical exclusive OR on the two values to the left and right of the operator when used with the VAR or VARI command. The Boolean Exclusive Or (^) performs a bitwise exclusive OR on the two values to the left and right of the operator when used with the VARB command.

For a logical exclusive OR (using VAR or VARI), the possible combinations are as follows:

```

positive number ^ positive number           = 0
positive number ^ zero or a negative number = 1
zero or negative number ^ positive number   = 1
zero or negative number ^ zero or negative number = 0

```

```

Example:  VAR1=5 ^ -1
Result:   VAR1=1

```

For a bitwise exclusive OR (using VARB), the value on the left side of the ^ operator has each of its bits exclusive *ORed* with the corresponding bit of the value on the right side of the operator. Each bit comparison will be composed of 9 possible combinations:

```

1 ^ 1 = 0           1 ^ X = X
1 ^ 0 = 1           X ^ 1 = X
0 ^ 1 = 1           0 ^ X = X
0 ^ 0 = 0           X ^ 0 = X
X ^ X = X

```

```

Example:  VARB1=b0000 1111 XXX1 ^ b10XX 10XX 10XX
Response to VARB1 is *VARB1=10XX_01XX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX

```

```

Example:  VARB1=h32FD ^ h6A
Response to VARB1 is *VARB1=1010_0001_1111_1011_0000_0000_0000_0000

```

```

Example:  VARB1=h7FFF ^ b1101 1111 0000 1101
Response to VARB1 is *VARB1=0011_0000_1111_0010_XXXX_XXXX_XXXX_XXXX

```

The total command length must be less than 80 characters. The order of precedence is **left to right**. The Operation Priority Level ( ) operators can be used; however, they cannot be nested.

| <b>[ ~( ) ]</b> |   | <b>Boolean Not</b> |            |
|-----------------|---|--------------------|------------|
| Type            | Operator (Bitwise)  | <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        | [ = ], [ & ], [ ^ ], [   ], [ << ], [ >> ], VAR, VARI, VARB |                    |            |

The Boolean Not (~) operator performs a logical NOT on the value immediately to its right when used with the VAR or VARI command. The Boolean NOT (~) performs a bitwise NOT on the value immediately to its right when used with the VARB command. Parentheses ( ) are required.

For a logical NOT (using VAR or VARI), the possible combinations are as follows:

```

~ (positive number)           = 0
~ (zero or a negative number) = 1

```

```

Example:  VAR1=~(5)           ; Result: VAR1=0

```

```

Example:  VAR1=~(-1)          ; Result: VAR1=1

```

For a bitwise NOT (using VARB), each bit is *NOTed*.

Example: VARB1=~(b0000 1000 1XX1)

Response to VARB1 is \*VARB1=1111\_0111\_0XX0\_XXXX\_XXXX\_XXXX\_XXXX\_XXXX

Example: VARB1=~(h32FD)

Response to VARB1 is \*VARB1=0011\_1011\_0000\_0100\_1111\_1111\_1111\_1111

The total command length must be less than 80 characters. The order of precedence is **left to right**.

The Boolean Not (~) operator also has one additional use. It can be used to change the sign of the distance (D) command. (e.g., if the distance has the values \*D+25000,+25000,+12000,-123000).

By issuing D~,~,~,~ the new values for distance would be \*D-25000,-25000,-12000,+123000.

---

|                     |  |  |                |            |
|---------------------|--|--|----------------|------------|
| <b>[ &lt;&lt; ]</b> |  | <b>Shift from R to L (Bit 32 to Bit 1)</b> |                |            |
| Type                | Operator (Bitwise)   |  | <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            | [ = ], [ & ], [ ^ ], [   ], [ ~ ], [ >> ], VAR, VARI, VARB |  |                |            |

---

The Shift R to L (<<) operator shifts a binary value from right to left (reducing its value) the number of bits specified. Zeros are shifted into the most significant bit locations. The number of bits to shift by is specified with the value immediately to the right of the (<<) operator, 32 maximum. The number of places to shift must be specified in either binary or hexadecimal format. (*The bits in the binary variable are displayed from 1 to 32, left to right, and shifting from right to left causes bits to be shifted from 32 to 1.*)

Example: VARB1=b0000 1000 1XX1 << b01

Response to VARB1 is \*VARB1=0010\_001X\_X1XX\_XXXX\_XXXX\_XXXX\_XXX\_XX00

Example: VARB1=b1111 0000 1111 << b001

Response to VARB1 is \*VARB1=0000\_1111\_XXXX\_XXXX\_XXXX\_XXXX\_XXXX\_0000

Example: VARB1= h0000 E3 << hA

Response to VARB1 is \*VARB1=0000\_0001\_1111\_0000\_0000\_0000\_0000\_0000

The total command length must be less than 80 characters. The order of precedence is **left to right**.

---

|                     |  |  |                |            |
|---------------------|--|--|----------------|------------|
| <b>[ &gt;&gt; ]</b> |  | <b>Shift from L to R (Bit 1 to Bit 32)</b> |                |            |
| Type                | Operator (Bitwise)   |  | <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            | [ = ], [ & ], [ ^ ], [   ], [ ~ ], [ << ], VAR, VARI, VARB |  |                |            |

---

The Shift L to R (>>) operator shifts a binary value from left to right (increasing its value) the number of bits specified. Zeros are shifted into the least significant bit locations. The number of bits to shift by is specified with the value immediately to the right of the (>>) operator, 32 maximum. The number of places to shift must be specified in either binary or hexadecimal format. (*The bits in the binary variable are displayed from 1 to 32, left to right, and shifting from left to right causes bits to be shifted from 1 to 32.*)

Example: VARB1=b0000 1000 1XX1 >> b01

Response to VARB1 is \*VARB1=0000\_0010\_001X\_X1XX\_XXXX\_XXXX\_XXXX\_XXXX

Example: VARB1=b1111 0000 1111 >> b001

Response to VARB1 is \*VARB1=0000\_1111\_0000\_1111\_XXXX\_XXXX\_XXXX\_XXXX

Example: VARB1= h45FA2 >> h4

Response to VARB1 is \*VARB1=0000\_0010\_1010\_1111\_0101\_0100\_0000\_0000

The total command length must be less than 80 characters. The order of precedence is **left to right**.

---

|          |  |                |            |
|----------|--|----------------|------------|
| <b>[</b> | <b>Send Response to Both Communication Ports</b> | <b>Product</b> | <b>Rev</b> |
| Type     | Communication Interface                          | 6K             | 5.0        |
| Syntax   | <!> [ <command><field1>                          |                |            |
| Units    | n/a  |                |            |
| Range    | n/a  |                |            |
| Default  | n/a  |                |            |
| Response | n/a  |                |            |
| See Also | BOT, PORT, ], ECHO, EOL, EOT, LOCK               |                |            |

---

The Send Response to All Ports ( [ ] ) command is used to send the response from the command which follows it to all communication ports. If a syntax error occurs, an error message will be sent to both communication ports.

**NOTE:** COM1 refers to the “RS-232” or “ETHERNET” connector, and COM2 refers to the “RS-232/485” connector.

**Example**

```
[TER          ;Transfer TER Status to both serial ports
```

---

|          |  |                |            |
|----------|--|----------------|------------|
| <b>]</b> | <b>Send Response to Alternate Communication Port</b> | <b>Product</b> | <b>Rev</b> |
| Type     | Communication Interface                              | 6K             | 5.0        |
| Syntax   | <!> ] <command><field1>                              |                |            |
| Units    | n/a  |                |            |
| Range    | n/a  |                |            |
| Default  | n/a  |                |            |
| Response | n/a  |                |            |
| See Also | BOT, PORT, [, ECHO, EOL, EOT, LOCK                   |                |            |

---

The Send Response to Alternate Port ( ] ) command is used to send the response from the command which follows it to the alternate port from the one selected. If a report back is requested from port COM1, the response will be sent out port COM2, and vice-versa. If a command is in a stored program, the report will be sent out the alternate port from the one selected by the PORT command. If a syntax error occurs an error message will be sent to the alternate port from the one selected.

**NOTE:** COM1 refers to the “RS-232” or “ETHERNET” connector, and COM2 refers to the “RS-232/485” connector.

**Example**

```
; *****
; In this example, we place the "]TAS" statement in a program so that
; we can select the port (in this case, "PORT1" selects COM1) as a
; reference. Otherwise, executing "]TAS" outside of a program merely
; sends the response to whatever port you are not communicating through.
; *****
DEF COM          ; Begin definition of program called "COM"
PORT1           ; Select COM1
TER             ; Transfer TER Status to port COM1
]TAS            ; Transfer TAS Status to port COM2
END             ; End program definition
```