DIRECT ATTACH

Format:

```
ATTACH pvect
ATTACH OFF{A/B/C}
ATTACH ?
```

Where: *pvect* is a vector.

Description:

`ATTACH pvect`

Attaches the specified position vector to the teach pendant according to the group for which the position vector is defined.

When a vector is attached to the teach pendant, all references to that group refer to the positions in the attached vector.

Only one vector at a time may be attached to each group. Attaching another position vector cancels the previous attachment for this group.

```
ATTACH OFFA
ATTACH OFFB
ATTACH OFFC
```

Detaches the position vector from teach pendant according to the group specified.

`ATTACH ?`

Displays the current ATTACH status.

Examples:

-
```
DIMP ALPHA[20]
ATTACH ALPHA
```

Defines a position vector for group A named ALPHA containing 20 positions.
Attaches vector ALPHA to teach pendant. A reference from teach pendant to position 15 will now actually refer to the position ALPHA[15].

-
```
DIMPB &BETA[30]
ATTACH &BETA
```

Defines a position vector for group B named &BETA containing 30 positions.
The & prefix enables this vector to be manipulated by the DELETE and INSERT commands.

- `ATTACH OFFB`

Detaches from the teach pendant the currently attached group B position vector.

AUTO DIRECT

Format: AUTO

Description: This command must be entered after the Auto/Teach switch on the teach pendant is moved from Teach to Auto.

AUTO transfers control from the teach pendant to the keyboard.

DIRECT/EDIT CLOSE

Format: `CLOSE [var]`

Where: *var* is a variable or constant, $0 \leq var \leq 5000$.

Description: The CLOSE command closes both an electric gripper and a pneumatic gripper.

`CLOSE`	Closes gripper until end of gripper motion.
`CLOSE var`	*Var* is the DAC value which is applied to the gripper motor to maintain drive for additional grasping force. The greater the value of *var*, the stronger the drive force. The DAC value is ignored when a pneumatic gripper is installed.

If the gripper is connected to the control loop, the CLOSE command disconnects it before executing the gripper motion.

Warning! Use the var *option with extreme caution to avoid damage to the motor and its gear. Use this command for brief periods, and set the DAC value as low as possible.*

Examples:

■ `CLOSE`	Closes gripper.
■ `CLOSE 1000`	Sets gripper DAC value to 1000.
■ `CLOSE PRESS`	Sets gripper's DAC value according to the value of variable PRESS.

Notes: Refer to the OPEN and JAW commands.

Refer to the section, "Peripheral Setup," in the ***ATS*** *Reference Guide*.

Also refer to the gripper parameters in Chapter 7.

DIRECT COFF

Format: `COFF[A/B]`

`COFF` *n*

Where: *n* is an axis in group C.

Description:

`COFF`	Disables servo control of all axes.
`COFFA` `COFFB`	Disables servo control of group A or group B.
`COFF` *n*	Disables servo control of a specific axis in group C.

When COFF is active, the axes cannot be operated. You must activate CON before motion can resume.

The COFF mode is activated when one of the following occurs:

- COFF is entered from the keyboard or Control Off is entered from the teach pendant.
- An EMERGENCY button is pressed. (After the button is released, CON must be entered to restore servo control.)
- The controller detects an impact or thermic error condition (as determined by parameter settings).

COFF must be activated before you change parameters values.

COFF must be activated if you want to move the axes by hand.

When the COFF mode is activated, the following message appears on both the computer screen and the teach pendant display:

```
CONTROL DISABLED
```

Examples:

- `COFF` Control OFF for all axes.
- `COFFA` Control OFF for group A axes.
- `COFF 10` Control OFF for axis 10 in group C.

Note: Refer to the CON command.

CON DIRECT

Format: `CON[A/B]`

`CON` *n*

Where: *n* is an axis in group C.

Description:

`CON`	Enables servo control of all axes.
`CONA` `CONB`	Enables servo control of group A or group B.
`CON` *n*	Enables servo control of a specific axis in group C.

When either CON (from keyboard) or Control On (from the teach pendant) is activated, the following message appears on both the computer screen and the teach pendant display:

```
CONTROL ENABLED
```

The controller must be in the CON state for axis operation.

Entering the command CON has no effect is the axis is already enabled.

Examples:

- `CON` Control ON for all axes.
- `CONA` Control ON for group A axes.
- `CONB` Control ON for group B axes.
- `CON 10` Control ON for axis 10 in group C.

Note: Refer to the COFF command.

DEFINE EDIT

Format: `DEFINE` *`var1`* `[`*`var2 ... var12`*`]`

Where: *var1* , *var2* , . . . *var12* are user variables.

Description: Defines a private variable. A private variable is recognized only by the specific program which was being edited when the DEFINE *var* command was entered.

Up to twelve variables can be defined in one command.

Examples:

- `DEFINE I` Creates a private variable named I.
- `DEFINE L ALL KEY` Creates private variables named L, ALL and KEY.

Note: This command does not create a program line.

EDIT DELAY

Format: `DELAY` *var*

Where: *var* is a variable or constant.

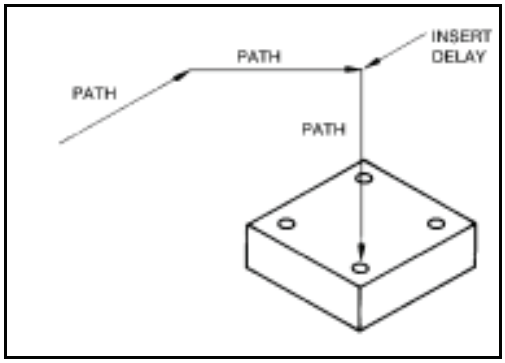
Description: Delays the execution of a program.

Var is defined in hundreths of a second (0.01 second).

The DELAY command is used for the following purposes:

- To insert a specific time delay between the execution of any two commands in a program.
- To enable the control system to stabilize at a certain position during the execution of movement commands. This compensates for differences in motion conditions (such as speed, direction, payload) between the time positions are recorded, and when they are approached at run-time.

The diagram here suggests a point for delaying the execution of a program, before the robot inserts a pin into a hole.



Examples:

- `DELAY 100` Delays for 1 second.
-
```
SET T=500
DELAY T
```
Delays for 5 seconds.

DELETE DIRECT

Format: `DELETE &pvect[n]`

Where: *&pvect* is a vector;
n is the index of one of the positions in the vector.

Description: Deletes position *n* in vector *&pvect*; all positions above position *&pvect*[*n*] are moved down one place until an empty (unrecorded) position is encountered.

The DELETE command can only be applied to a position vector whose name begins with the character *&*.

DELETE can only be applied to a position for a robot or multi-axis device which is dedicated to group A or group B. The command is not applicable to positions for a single axis device.

You can delete a position only if it is not used by any program in the user RAM. The controller will warn you if you attempt to delete a position which is in use.

Example:

- `DELETE &AA[4]` Deletes position 4 in vector &AA.

Note: Refer to the DIMP and INSERT commands.

DELP

Format:

DELP *pos*

DELP *pvect*

Where: *pos* is a position;
pvect is a vector.

Description:

Deletes positions and position vectors from user RAM.

You can delete a position or vector only if it is not used by any program in the user RAM. A warning will appear if you attempt to delete a position which is in use.

The DELP command cannot delete individual positions within a vector.

The DELETE command can delete individual positions within a vector, providing the vector name has the prefix &.

The UNDEF command erases the coordinate values of a position, but keeps the position defined.

Examples:

- `DELP A9` — Deletes a position or a vector named A9.
- `DELP DOD` — Deletes a position or a vector named DOD.
- `DELP &BB` — Deletes a vector named &BB.

Notes:

This command does not create a program line.

Refer to the DELETE and UNDEF commands.

DELVAR DIRECT/EDIT

Format: DELVAR *var*

Where: *var* is a user variable or variable array.

Description: Erases a user defined variable or variable array from user RAM.

You can delete a variable only if it is not used by any program in the user RAM. A warning will appear if you attempt to delete a variable which is in use.

In DIRECT mode DELVAR deletes only global variables.

In EDIT mode DELVAR deletes private variables; it will delete a global variable only if a private variable with that name does not exist.

You cannot delete system variables.

Examples:

- `DELVAR X` Deletes variable X.
- `DELVAR PRESS` Deletes variable PRESS.

Note: This command does not create a program line.

Format: `DIM var[n]`

Where: *var* is a user defined name;
[*n*] is the dimension of the array.

Description: Defines a private variable array of *n* elements. The elements created are named *var*[1], *var*[2], . . . *var*[*n*].

A private variable is recognized only by the program it which it is defined.

Example:

- `DIM PRGV[20]` — Creates a variable array named PRGV containing 20 private variables, PRGV[1] . . . PRGV[20].

Note: This command does not create a program line.

DIMG DIRECT/EDIT

Format: `DIMG var[n]`

Where: *var* is a user defined name;
[*n*] is the dimension of the array.

Description: Defines a global variable array of *n* elements. The elements created are named *var*[1], *var*[2] . . . *var*[*n*] .

A global variable can be used by any user program.

Example:

- `DIMG GLOB[8]` — Creates a variable array named GLOB containing 8 global variables, GLOB[1] . . . GLOB[8].

Note: This command does not create a program line.

DIMP

Format:

`DIMP[A/B] pvect[n]`

`DIMPC pvect[n] axis`

Where: *pvect* is a user defined name;
[*n*] is the dimension of the vector;
axis is an axis in group C.

Description:

Defines a vector containing *n* positions, named *pvect*[1], *pvect*[*2*], . . . *pvect*[*n*], for a specific axis control group. When a vector is defined, controller memory is reserved for the coordinate values of the positions which will subsequently be recorded or set.

`DIMP pvect[n]` `DIMPA pvect[n]`	Defines a vector for axis control group A.
`DIMPB pvect[n]`	Defines a vector for axis control group B.
`DIMPC pvect[n] axis`	Defines a vector for an axis in group C.

If a group is not specified for the vector, group A is assumed. Once a vector has been defined, it is dedicated to a specific axis control group, and cannot be used to record coordinates for a different axis control group.

The first character of the vector name must be a letter or the character &.

When the first character of the vector name is &, the vector can be manipulated by the DELETE and INSERT commands. Include the & prefix in the vector name if you intend to use the vector for SPLINE or MOVES trajectories.

Examples:

- `DIMP PICK[30]` — Creates a vector for group A containing 30 positions named PICK[1] . . . PICK[30].
- `DIMPB &BETA[30]` — Creates a vector for group B containing 30 positions, named &BETA[1] . . . &BETA[30]. The & prefix enables this vector to be manipulated by the DELETE and INSERT commands.
- `DIMPC CNV[25] 11` — Creates a vector for axis 11 containing 25 positions named CNV[1] . . . CNV[25].

Notes:

This command does not create a program line.

DIR DIRECT

Format: DIR

Description: Displays a list of all current user programs. The four columns provide the following information:

- Program Name.
- Program Validity.
  If the program contains a logic error, NOT VALID will be displayed.
- Program Identity Number.
  This is a controller assigned program number; this is the number you need to use for accessing programs from the teach pendant.

  Since certain controller operations will cause the ID number to change, it is recommended that you use the DIR command at the beginning of each working session to verify the ID numbers of the programs you will want to run from the teach pendant.
- Program Execution Priority.

Example:

```
>DIR
name          : validity   : identity : priority
IO            :            : 2          : 5
IOA :         : 3          : 5
TWOIO         :            : 4          : 5
INOUT         :            : 5          : 5
PICP          :            : 6          : 5
```

Notes: Validity: Refer to the EXIT command.

Priority: Refer to the PRIORITY and RUN commands.

ELSE

Format: `ELSE`

Description: The ELSE command follows an IF command and precedes ENDIF.

ELSE marks the beginning of a program subroutine which defines the actions to be taken when an IF command is false.

Example:

```
IF J>2                  If the value of J is greater than 2,
    ANDIF A=B           and if the values of A and B are equal,
    SET OUT[1]=1        the controller will turn on output 1;
ELSE                    If any of the conditions is not true,
    SET OUT[5]=1        the controller will turn on output 5.
ENDIF
```

Note: Refer to the IF command.

END EDIT

Description: The system automatically writes END as the last line of a program.
The system automatically writes (END) at the end of a listing.
END is not a user command.

ENDFOR

Format: `ENDFOR`

Description: Required companion to FOR command.

Ends the subroutine to be executed by the FOR command.

Example:

```
FOR I=1 TO 16
    SET OUT[I]=1
ENDFOR
```

This loop is performed 16 times, and turns on all 16 outputs

Note: Refer to the FOR command.

ENDIF EDIT

Format: `ENDIF`

Description: Required companion to IF command.

Ends the subroutine to be executed by the IF command.

Example:

```
IF XYZ=1                    If the first condition and the second condition are
   ANDIF Z[1]=X             true,
   ORIF B<C                 or if the third condition is true,
   MOVE POS[1]              execute the move;
ELSE                        otherwise,
   MOVE POS[2]              execute a different move.
ENDIF
```

Note: Refer to the IF command.

<Enter> EDIT/DIRECT

Format: `<Enter>`

Description: In EDIT mode, checks the syntax of the line, then goes to the next line in program and displays its number.

In DIRECT mode, confirms and executes the command.

The following **ACL** commands do not require <Enter> for execution:

`<Ctrl>+A`	Immediately aborts all running user programs and stops axes movement.
~	Toggles Manual Keyboard mode on and off.
`<Ctrl>+C`	Cancels the display of data resulting from SHOW ENCO, LIST, SEND, and other commands.

Format: `FOR var1=var2 TO var3`

Where: *var1* is a variable;
var2 and *var3* are variables or constants.

Description: Executes a subroutine for all values of *var1*, beginning with *var2* and ending with *var3*.

The last line of the subroutine must be the ENDFOR command.

Examples:

-
```
FOR L=M TO N
   MOVED POS[L]
ENDFOR
```

-
```
FOR I=1 TO 16
   SET OUT[I]=1
ENDFOR
```

GET EDIT

Format: `GET var`

Where: *var* is a user variable.

Description: When the program encounters a GET command, it pauses and waits for a keyboard character to be pressed. The variable is assigned the ASCII value of the character that is pressed.

The GET command should be preceded by a PRINTLN command which will indicate to the user that the program is waiting for a character to be pressed.

Example:

```
PRINTLN "SELECT PROGRAM: P Q R"
GET VP                    (VP is the variable)
IF VP=80                  (80 is ASCII for P)
   ORIF VP=112            (112 is ASCII for p)
   RUN P
ENDIF
IF VP=81                  (81 is ASCII for Q)
   ORIF VP=113            (113 is ASCII for q)
   RUN Q
ENDIF
IF VP=82                  (82 is ASCII for R)
   ORIF VP=114            (114 is ASCII for r)
   RUN R
ENDIF
```

Note: Refer to the READ command.

GLOBAL DIRECT/EDIT

Format: `GLOBAL` *`var1`* `[`*`var2 ... var12`*`]`

Where: *var1* [*var2 . . .var12*] are user defined variables.

Description: Defines a global variable. A global variable can be used in any user program. Up to twelve variables can be defined in one command.

Examples:

- `GLOBAL HB` Creates a global variable named HB.
- `GLOBAL J BYE ME` Creates global variables named J, BYE and ME.

Note: This command does not create a program line.

Format: `GOSUB prog`

Where: *prog* is a user program.

Description: Transfers program control from the main program to *prog*, starting at the first line of *prog*. When the END command in *prog* is reached, execution of the main program resumes with the command which follows the GOSUB command.

Example:

```
SET Z=10
GOSUB SERVE
MOVE P3
```

After executing the SET command, and before executing the MOVE command, the program SERVE is executed in its entirety.

GOTO EDIT

Format: `GOTO labeln`

Where: *labeln* is any number, $0 \leq n \leq 9999$

Description: Jumps to the line immediately following the LABEL *labeln* command.

LABEL *labeln* must be included in the same program as the GOTO command.

Examples:

- This program is executed in an endless loop unless aborted manually.

```
LABEL 5
MOVE POS13
SET A=B+C
GOSUB MAT
GOTO 5
```

- This program is executed 500 times and then stops.

```
LABEL 6
GOSUB BE
SET K=K+1
IF K<500
   GOTO 6
ENDIF
```

Note: Refer to the LABEL command.

HELP

Format: `HELP [topic]`

Description: When in DIRECT mode, HELP provides an on-line help screen for DIRECT commands, as follows:

`HELP`	A list of topics is displayed on your screen. ACL commands appear in uppercase letters; other subjects appear in lowercase letters.
`HELP topic`	Where *topic* is the name of a command or subject. A brief explanation of the topic is displayed on your screen.
`DO HELP`	Provides on-line help screen for EDIT commands.

When in EDIT mode, HELP provides a list and brief explanation of all EDIT commands.

This command does not create a program line.

HERE / HEREC DIRECT/EDIT

Format:

`HERE` *pos*

Where: *pos* is a position for any axis group.

`HEREC` *pos*

Where: *pos* is a robot (group A) position.

Description: Records an absolute position, according to the current location of the axes.

`HERE` *pos*	Records in joint (encoder) values the current coordinates of the axes for the specified position.
`HEREC` *pos*	Records in Cartesian (world) coordinate values the current coordinates of the axes for the specified position. This command is valid for robot (group A) axes only.

If the position has an alphanumeric name, it must first be defined using the DEFP or DIMP command.

The DEFP command is not required if the position is a *numerically* named position for group A; it will be automatically defined when entered as part of the HERE/HEREC command.

Examples:

- `HERE 3` — Defines and records the coordinates of position 3 for group A.

- `DEFPB POINT`
`HERE POINT` — Defines a position named POINT for group B; records the current coordinates for position POINT.

- `DIMP P[20]`
`HEREC P[5]` — Defines a vector named P containing 20 position for group A; records Cartesian coordinates for position 5 in the vector.

- `MOVE POSA 300`
`DELAY 100`
`HERE PMID` — One second after movement to position POSA begins, an intermediate position, PMID, is recorded according to the axes' location at that moment.

Format: `IF var1 oper var2`

Where: *var1* is a variable;
var2 is a variable or constant;
oper can be: `<`, `>`, `=`, `<=`, `>=`, `< >`

Description: The IF command checks the relation between *var1* and *var2* .
If it meets the specified conditions, the result is true, and the next sequential program line is executed (subroutine or command). If it is not true, another subroutine or command is executed.

Examples:

-
```
IF C[1]=3
    MOVE AA[1]
ELSE
    GOSUB TOT
ENDIF
```
If C[1] = 3, then move to AA[1]. If C[1] ≠ 3 , execute (subroutine) program TOT.

-
```
IF IN[3]=1
    SET OUT[7]=1
ELSE
    MOVE 10
ENDIF
```
If input 3 is on, controller will turn on output 7; if input 3 is off, robot will move to position 10.

-
```
IF A > 5
    GOSUB WKJ
ENDIF
```
If variable A is greater than 5, (subroutine) program WKJ will be executed.

Note: Refer to the commands ELSE, ANDIF, ORIF, and ENDIF.

JAW DIRECT/EDIT

Format: `JAW var [duration]`

Where: *var* and *duration* are variables or constants.

Description: *Var* is the size of the gripper opening, defined as a percentage of a fully opened gripper.

Duration is defined in hundreths of a second.

The JAW command can be used only for servo grippers.

The JAW command brings the gripper opening to size *var* within the specified time. If *duration* is omitted, movement is at maximum speed.

Warning! Be sure you select a proper value for the gripper opening. An incorrect size will cause constant and excessive power to motor, and may damage the motor.

The JAW command activates servo control for the gripper axis, while the OPEN/CLOSE commands disconnect the gripper axis from the servo control loop.

Unless you need the JAW command for a specific application, the OPEN and CLOSE commands are recommended.

Examples:

- `JAW 40` Brings the gripper to 40% of full opening.
- `JAW 0 300` Closes the gripper in 3 seconds.

Note: Refer to the CLOSE and OPEN commands.

LABEL EDIT

Format: `LABEL` *`labeln`*

Where: *labeln* is any number, 0 ≤ *labeln* ≤ 9999.

Description: Marks the beginning of a program subroutine which is executed when the GOTO command is given.

Example:

```
LABEL 12
MOVEL 1
MOVE 15 200
OPEN
MOVE JJ
GOTO 12
```

Note: Refer to the GOTO command.

LIST DIRECT

Format: `LIST [prog]`

Where: *prog* is a program.

Description:

`LIST`	Displays all lines of all programs.
`LIST prog`	Displays all lines of program *prog*.
`LIST prog > PRN:`	Prints the specified program at a printer connected to a parallel communication port.

When using the LIST command to view program lines, the commands ENDFOR, ENDIF and ELSE are followed by the line number of the corresponding FOR and IF commands.

Example: ■ Displays all lines in program AAA.

```
>LIST AAA
   PROGRAM   AAA
   *****************
25:  LABEL 1
26:  MOVED 31
27:  MOVED 32
28:  IF IN[3]=1
29:     SET OUT[7]=1
30:  ELSE   (28)
31:     SET OUT[5]=1
32:  ENDIF  (28)
33:  MOVED 33
34:  GOTO   1
35:  END
(END)
```

Note: Refer to the SEND commands.

Format: LISTP

Description: Displays a list of all defined positions with alphanumeric names, and the group to which they are dedicated.

Positions with numeric names are not listed.

Example:

```
>LISTP
   DEFINED POINTS
   **************

point name: group       :(axis)
- - - - - - - - - - - - - - - - - - - - - - - -
P[10]                   : A
PICP[10]                : A
AA          : A
B1                      : B
B2                      : B
BBA[50]                 : B
C1          : C         : 10
C2          : C         : 11
C3[100]                 : C         : 11

and 13 additional numerical points defined.
```

LISTPV DIRECT

Format:

```
LISTPV pos
LISTPV POSITION
```

Description:

`LISTPV` *pos* — Displays in joint (encoder) values the coordinates of the specified position. If *pos* is a robot (group A) position, joint and Cartesian coordinates are both displayed.

`LISTPV POSITION` — Displays the current coordinates of the robot arm. POSITION is a position name reserved by the system for the current position of the robot (group A) axes.

Displays the type and coordinates of a recorded position, or POSITION:

- Position name, and type of position (one of the following):
  - Joint position
  - World A position (Cartesian coordinates in area A)
  - World B position (Cartesian coordinates in area B)
  - Relative by axis to CURRENT position.
  - Relative by XYZ to CURRENT position.
  - Relative by axis to position *PNAME*.
  - Relative by XYZ to position *PNAME*.
- Joint values of the position: encoder counts for each axis.
- Cartesian coordinates of a robot (group A) position only; the distance from the robot's point of origin— the center and bottom of the robot's base—to the TCP (tool center point). X, Y, and Z are displayed in millimeters, accurate to a micron (thousandth of a millimeter). Pitch and roll (P and R) values are displayed in degrees with an accuracy of 0.002°.

Example:

```
>LISTPV POS1
Position POS1 (Joint)
1: 0        2: 5        3: 13926    4: 0        5: 0
X: 5.425    Y: 0.000    Z: 29.963   P: 12.640   R: -3.200
```

DIRECT LISTVAR

Format: LISTVAR

Description: Displays a list of all user and system variables.

Variable arrays include an index in square brackets, which indicates the dimension of the array; for example, IN[16].

Private variables include (in parentheses) the name of the program to which they are dedicated; for example, I(INOUT).

Example:

```
>LISTVAR

   SYSTEM VARIABLES
   ****************
IN[16]
ENC[12]
HS[12]
ZP[12]
CPOS[12]
TIME
LTA
LTB
MFLAG
ERROR
ERRPR
ERRLI
OUT[16]
ANOUT[12]

   USER VARIABLES
   **************
I(DEMO)
J(DEMO)
I(IO)
I(INOUT)
G1
G2
```

Note: Refer to Chapter 4 for a description of system variables.

Note that execution of MOVE is not synchronized with program flow. MOVED is usually more suitable for most applications.

Format:

`MOVE` *`pos`* [*`duration`*]

`MOVED` *`pos`* [*`duration`*] EDIT mode only.

Where: *pos* is a position;
duration is a variable or a constant.

MOVE

Description:

`MOVE` *`pos`*	Moves the robot to the specified position, according to the speed defined by a preceding SPEED command.
`MOVE` *`pos duration`*	Moves the robot to the position within the specified amount of time. *Duration* is defined in hundredths of a second.

The MOVE command deposits a movement command into the **movement buffer**. The program issuing the MOVE command does not wait for the operation to be completed, and continues regardless of when the MOVE command is executed.

If the program contains several consecutive MOVE commands, they are sent until the movement buffer is full, regardless of the actual execution. As a result, program commands other than MOVE may not be executed according to the intended sequence.

MOVE is executed according to speed (SPEED) or time (*duration*), regardless of how accurately the axes reach the target position.

To ensure sequentiality in a program, do one of the following:

- Use the MOVE with its *duration* (*time*) option, followed by a DELAY command of equal duration. For example:

```
MOVE pos1 time1
DELAY time1
MOVE pos2 time2
DELAY time2
```

- Use sequencing commands, such as WAIT. For example:

```
MOVE pos1
WAIT IN[1]=1
```

- Use the MOVED command.

MOVED

Description: The MOVED command ensures that operations defined in the program are executed sequentially.

A MOVED command is deposited into the movement buffer only when the previous MOVED command has been completely executed.

A MOVED command is terminated only when the axes have arrived at their target position within the specified accuracy, no matter how long it takes, and even when *duration* has been defined.

To ensure that the MOVED is executed within a defined period of duration, issue the EXACT OFF command. For example:

```
EXACT OFFA
MOVED POS1 500
MOVED POS2 500
```
Axes reach POS1 and POS2 in 5 seconds.

```
EXACT A
MOVED POS3
```
Axes reach POS3 with required accuracy, regardless of duration.

MOVE, MOVED Summary

`MOVE`	Easy to program, but cannot guarantee sequentiality or accuracy.
`EXACT MOVED`	Guarantees sequentiality and accuracy, but not duration.
`EXACT OFF MOVED`	Guarantees sequentiality and duration, but not accuracy.

Examples:

-
```
MOVE 3
MOVE AA
PRINT "COMMAND GIVEN"
```
The robot moves to position 3 and then to position AA. The line "COMMAND GIVEN" will probably be displayed before actual movement is completed.

-
```
MOVE 3
MOVE AA
MOVE POS[1]
SET OUT[1] = 1
DELAY 1000
```
The three movement commands are deposited almost simultaneously in the movement buffer. The robot moves to position 3, then to AA and then to POS[1]. Concurrent with the movement to position 3, output 1 is turned on, and the program is delayed for 10 seconds. This program ends about 10 seconds after its activation, regardless of the axes' location.

-
```
MOVE 3 500
DELAY 500
MOVE AA 800
DELAY 800
MOVE POS[1] 200
DELAY 200
SET OUT[1]=1
DELAY 1000
```
The robot moves to position 3 in 5 seconds, then to AA in 8 seconds, then to POS[1] in 2 seconds. Then output 1 is turned on, and a delay of 10 seconds occurs. Total time for program execution is 25 seconds, plus a negligible fraction of time for command executions.

-
```
MOVED 3
SET OUT[1]=1
DELAY 1000
MOVED AA
MOVED POS[1]
```
All the commands are executed in sequence. All positions are accurately reached. The axes will pause at some of the positions.

-
```
EXACT OFFA
MOVED 3
MOVED AA
EXACT A
MOVED POS[1]
CLOSE
SET OUT[1]=1
```
This program format is recommended, assuming that positions 3 and AA are along a path, and position POS[1] is where an object is picked up. Position 3 and AA are reached in specified time, regardless of accuracy. Position POS[1] is accurately reached, but with a possible delay. All commands in this program are activated in sequence.

Note: Refer to the EXACT command.

MOVEC / MOVECD DIRECT/EDIT

Format:

`MOVEC` *`pos1 pos2`*

`MOVECD` *`pos1 pos2`* EDIT mode only.

Description:

Moves the robot's TCP (tool center point) along a **circular** path, from its current position to *pos1,* through *pos2* .

The coordinates of *pos2* and *pos1* determine the length of the path.
A preceding SPEEDL command defines the speed of the TCP. The duration of the movement is thus determined by the path length and the SPEEDL definition.

The starting position, *pos1*, and *pos2* should define a circle. These three points should not be aligned, and should have different coordinates.

MOVEC/MOVECD is executed in the Cartesian coordinate system, and is only valid for robot (group A) axes.

All other aspects of the MOVEC/MOVECD commands are similar to those of the MOVE/MOVED commands.

Warning! Be careful when recording positions for MOVEC commands. Mechanical limitations or obstacles, such as the robot itself, may make the resulting path invalid.

Examples:

- `MOVEC 1 2` — Moves along a circular path from current position to position 1 via position 2.

- `SPEEDL 20`
  `MOVEC 2 1` — Moves along a circular path from current position to position 2 via position 1, at a speed of 20mm per second.

Note:

Refer to the SPEEDL command.

MOVEL / MOVELD

Format:

`MOVEL` *`pos1`* [*`duration`*]

`MOVELD` *`pos1`* [*`duration`*] EDIT mode only.

Description:

Moves the robot's TCP (tool center point) along a **linear** path (straight line) from its current position to *pos1*.

If duration is not specified, the speed of the TCP is defined by a preceding SPEEDL command.

MOVEL/MOVELD is executed in the Cartesian coordinate system, and is only valid for robot (group A) axes.

All other aspects of the MOVEL/MOVELD commands are similar to those of the MOVE/ MOVED command.

Warning! Be careful when recording positions for MOVEL commands. Mechanical limitations or obstacles, such as the robot itself, may make the resulting path invalid.

Example:

- `MOVELD TR` Moves along a straight line to position TR.

Note:

Refer to the SPEEDL command.

MOVES / MOVESD DIRECT/EDIT

Format:

```
MOVES pvect n1 n2 [duration]
MOVESD pvect n1 n2 [duration]          EDIT mode only
```

Where: *pvect* is the name of position vector;
n1 is the index of the first position;
n2 is the index of the last position to be reached.

Description:

Moves the axes through any number of consecutive vector positions, from *n1* to *n2*, without pausing. The trajectory is calculated by a linear interpolation algorthim, then smoothed according to parameter 219.
(PAR 219 value is set on scale 1–200; 1=no smoothing; 200=smoothest.)

All positions in the vector must be absolute joint positions.

The duration of movement between any two consecutive positions is constant. The greater the distance between two consecutive vector positions, the faster the robot moves through that segment of the path. It is therefore recommended that vector positions be evenly spaced to allow a smooth movement.

If *duration* is not specified, the average speed of movement is determined by a preceding SPEED command.

MOVES/MOVESD can be executed only by a robot or multi-axis device, using group A or group B positions. The command is not applicable for a single axis device.

All other aspects of the MOVES/MOVESD commands are similar to those of the MOVE/MOVED commands.

Example:

`MOVED PATH[1]`	Moves to starting position PATH[1].
`MOVESD PATH 2 20`	Moves in a continuous path through positions PATH[2] to PATH[20].
`MOVESD PATH 19 1`	Then moves along the same path in the opposite direction.

Note:

Refer to the SPLINE and SPLINED commands.

DIRECT/EDIT OPEN

Format: `OPEN [var]`

Where: *var* is a variable or constant, $0 \le var \le 5000$.

Description: The OPEN command opens both an electric gripper and a pneumatic gripper.

`OPEN`	Opens gripper until end of gripper motion.
`OPEN var`	*Var* is the DAC value which is applied to the gripper motor to maintain drive for additional grasping force. The greater the value of *var*, the stronger the drive force. The DAC value is ignored when a pneumatic gripper is installed.

If the gripper is connected to the control loop, the OPEN command disconnects it before executing the gripper motion.

Warning! Use the var *option with extreme caution to avoid damage to the motor and its gear. Use this command for brief periods, and set the* var *value as low as possible.*

Examples:

■ `OPEN`	Opens gripper.
■ `OPEN 1000`	Sets gripper DAC value to 1000.
■ `OPEN PRESS`	Sets gripper DAC value according to the value of variable PRESS.

Notes: Refer to the CLOSE and JAW commands.

Refer to the section, "Peripheral Setup," in the ***ATS*** *Reference Guide.*

Also refer to the gripper parameters in Chapter 7.

ORIF EDIT

Format: `ORIF var1 oper var2`

Where: *var1* and *var2* are variables or constants;
oper can be: `<`, `>`, `=`, `<=`, `>=`, `< >`.

Description: An IF type command, ORIF logically combines a condition with other IF commands.

Example:

```
IF A=B                  If either A = B
   ORIF A=D             or A = D,
   CLOSE                close the gripper;
ELSE                    otherwise,
   OPEN                 open the gripper.
ENDIF
```

Note: Refer to the IF command.

PRINT

Format: `PRINT arg1 [arg2 ... arg4]`

Where: *arg* is a variable or a string within quotation marks (“ ”).

Description: Displays strings and variable values on screen.

The text following PRINT may contain up to 40 characters and spaces, not including the quotation marks. The text may contain a total of 4 arguments and/or variables.

A variable is one argument, regardless of length.

A string of up to ten characters is one argument. Strings which exceed 10, 20 and 30 characters are treated, respectively, as two, three and four arguments.

Example:

```
SET NA=5
PRINT "THE ROBOT HAS " NA " AXES"
```

Will display on screen:

```
THE ROBOT HAS 5 AXES
```

The text THE ROBOT HAS is arguments 1 and 2 (contains 13 characters);
the variable NA is argument 3;
the text AXES is argument 4.

Note: Refer to the PRINTLN command.

PRINTLN EDIT

Format: `PRINTLN arg [arg2 ... arg4]`

Where: *arg* is a variable or a string within quotation marks (" ").

Description: Displays strings and variable values on screen.

Same as PRINT command, but inserts a carriage return (to beginning of line) and a line feed (to next line) before the displayed text.

The text following PRINTLN may contain up to 40 characters and spaces, not including the quotation marks. The text may contain a total of 4 arguments and/or variables.

A variable is one argument, regardless of length.

A string of up to ten characters is one argument. Strings which exceed 10, 20 and 30 characters are treated, respectively, as two, three and four arguments.

Entering PRINTLN without an argument simply enters a carriage return and a line feed.

Example: ■

```
SET X=7
SET Y=15
SET J=8
SET K=20
PRINTLN "TANK # " X " LEVEL IS: "Y
PRINT " INCHES"
PRINTLN "TANK # " J " LEVEL IS : "K
PRINT " INCHES"
```

Will display:

```
TANK #7 LEVEL IS: 15 INCHES
TANK #8 LEVEL IS: 20 INCHES
```

Note: Refer to the PRINT command.

READ EDIT

Format: `READ` *arg1* [*arg2 ... arg4*]

Where: *arg* is a variable or a string within quotation marks (" ").

Description: When READ encounters an argument which is a **string**, the text will be displayed like a PRINT statement.

When READ encounters an argument which is a **variable**, a "?" will be displayed on screen, indicating that the system is waiting for a value to be entered.

The READ procedure is performed sequentially for all the arguments.

Your reply to "?" must be a numeric value. Pressing <Enter> without specifying a value will enter a value of 0.

Any other reply to "?" is interpreted as a command. If you enter a command, it will be executed, and the READ command will again prompt you to enter a value by displaying the message:

```
ENTER value >>
```

Example:

- `READ "enter value of x" X`

  Will display on screen:

  `enter value of x ?`

  If you enter 254, the value 254 will be assigned to variable X.

Note: Refer to the PRINT command.

REMOVE DIRECT

Format: `REMOVE` *`prog`*

Description: Deletes a user program from the user RAM and frees all memory allocated to that program.

The system will prompt for verification:

```
Are you sure? (yes/no)
```

To confirm, respond by typing YES (complete word).

Any response other than YES (including Y) will be interpreted as NO.

If program *prog* is called or used by other programs, the REMOVE is not allowed, and a list of all program lines referring to *prog* is displayed.

Private variables assigned to this program are also deleted.

Use the EMPTY command if you want to delete all program lines without deleting the program itself.

Example:

- `REMOVE PALET` Deletes program PALET.

Note: Refer to the EMPTY command.

RUN DIRECT/EDIT

Format: `RUN prog [var]`

Where: *prog* is a program;
var is a variable or constant.

Description: Starts execution of a task from the first line of program *prog* .

Var is the priority of the program, and ranges 1 to 10; 10 is the highest priority. If the value of *var* is greater than 10, priority is set to 10.
If the value of *var* is less than 1, priority is set to 1. By default (when controller is powered on), all programs are assigned a priority of 5.

When a running program encounters a RUN *prog* command, both programs are executed concurrently. If several programs are activated, those with a higher priority are executed first. Programs with equal priority run concurrently; these programs share CPU time by means of an equal distribution algorithm.

In EDIT mode, if priority is not specified in the RUN command, the program's priority is automatically set to a default value of 5.

In DIRECT mode, if priority is not specified in the RUN command, the program's priority is set to the value last defined by a preceding PRIORITY or RUN command.

Examples:

-
```
>PRIORITY 10
>RUN DEMO
>RUN PLT
```
Programs DEMO and PLT run at the highest priority.

- `RUN DEMO` — Program DEMO runs at default priority 5.
- `RUN IOS 9` — Program IOS runs with a priority value of 9.

Note: Refer to the PRIORITY command.

Format:

```
SET var1=var2
SET var1=oper var2
SET var1=var2 oper var3
SET var1=COMPLEMENT var2
SET var=PVAL pos axis
SET var=PVALC pos coord
SET var=PSTATUS pos
SET var=PAR n
```

Where: *var* and *var1* is a variable;
var2 and *var3* can be either a variable or a constant.

oper can be:
Arithmetic operator: + – * /
Algebraic operator: ABS, EXP, LOG, MOD
Trigonometrical operator: COS, SIN, TAN, ATAN
Logical (Boolean) operator: AND, OR, NOT

pos is a position;
axis is an axis number;
coord is a Cartesian coordinate: X, Y, Z, or P or R;
n is a parameter number.

Description:

1.	`SET var1=var2`	Assigns the value of *var2* to *var1*.
2.	`SET var1=oper var2`	The operation is performed on *var2* and the result is assigned to *var1* .
	If *oper* is ABS	Assigns the absolute value of *var2* to *var1*
	If *oper* is NOT	Assigns the logical negative value of *var2* to *var1*. If *var2* ≤ 0, *var1* = 1; If *var2* >0, *var1* =0.

3. `SET var1=var2 oper var3`

If *oper* is : +, -, *, /, MOD	The operation is performed on *var2* and *var3* and the bitwise result is assigned to *var1* .
If *oper* is: AND, OR	The binary operation is performed on *var2* and *var3* and the result is assigned to *var1*.
If *oper* is: COS, SIN, TAN	*The controller uses integer arithmetic; fractional values are therefore scaled in order to produce accurate results.* Since the result of these trigonometric functions is always in the range of -1 to 1, the function of *var3* is computed and then multiplied by *var2*. (*Var2* must be large enough to give the expected accuracy.) The value of *var3* is an expression of degrees.
If *oper* is: ATAN, EXP, LOG	In order to use a practical value for *var3*, *var3* is first divided by 1000; then the function is applied. The result is then multiplied by *var2*. In **Controller-B**, the result of the ATAN function is an expression of degrees.

4. `SET var1=COMPLEMENT var2`

   Each individual bit of the binary representation of *var2* is inverted, and the result is assigned to *var1* .

5. `SET var=PVAL pos axis` Assigns *var* the joint value of the specified axis in the specified position. (Refer to the PVAL command.)

6. `SET var=PVALC pos coord`

   Assigns *var* one of the Cartesian coordinates of the specified robot (group A) position. (Refer to the PVALC command.)

7. `SET var=PSTATUS pos` Assigns *var* a value according to the type of the specified position. (Refer to the PSTATUS command.)

8. `SET var=PAR n` Assigns *var* the value of the specified parameter.

Examples:

■ `SET A=B`	Assigns value of B to A.
■ `SET A=NOT B`	If B is 0 then A is set to 1.
■ `SET A=COMPLEMENT B`	If B is 0 then A is set to -1.
■ `SET A=ABS B`	If B is -1 then A is set to 1.
■ `SET A=B AND C`	If B=1 and C=0, then A is set to 0.
■ `SET A=1000 COS 60`	COS 60 = .5; Multiply by 1000; A is set to 500.
■ `SET ST=PSTATUS P1`	If P1 is an absolute Joint position, then ST will be assigned a value of 1.
■ `SET XC=PVALC POS1 X`	XC receives the value of the robot's X-coordinate position POS1.
■ `SET A=PAR 76`	The value of parameter 76 is assigned to variable A.
■ `SET ANOUT[3]=2500`	**Available in PRIVILEGE mode only**. Sets the analog output value for axis 3 to 2500. (ANOUT[*n*] is a system variable.)
■ `SET OUT[5]=1`	Turns on output 5. (OUT[*n*] is a system variable.)
■ `SET CLOCK=TIME`	Assigns value of system variable TIME to user variable CLOCK.

SPEED DIRECT/EDIT

Format: `SPEED[A/B] var`

`SPEEDC var axis`

Where: *var* is a variable or constant.

Description: SPEED or SPEEDA sets the speed of group A axes.
SPEEDB sets the speed of group B axes.
SPEEDC sets the speed of a specific axis in group C.

Defines the speed of MOVE, MOVES, and Joint SPLINE movements in percentages. Maximum speed is 100; minimum is 1. The default speed is 50.

Movement commands which do not include a *duration* argument are executed according to the SPEED setting.

In DIRECT mode, the SPEED command takes effect immediately. Determines the speed of movement when the MOVE(D), MOVES(D) and Joint SPLINE(D) commands are executed in DIRECT mode.

In EDIT mode, the SPEED command takes effect after it is executed from within a program. Determines the speed of movement when the MOVE(D), MOVES(D) and Joint SPLINE(D) commands are executed from within a program.

To view current speed settings, use the SHOW SPEED command.

Examples:

- `SPEED 20` — Sets speed of joint movements of group A to 20% of maximum speed.
- `SPEEDB 50` — Sets speed of joint movements of group A to 50% of maximum speed.

Note: Refer to the MOVE(D), MOVES(D), SPLINE(D), SPEEDL and SHOW SPEED commands.

DIRECT/EDIT SPEEDL

Format: `SPEEDL` *var*

Where: *var* is a variable expressed in microns,
or a constant value expressed in millimeters.

Description: SPEEDL sets the speed of robot (group A) axes only.

Defines the speed of linear and circular (MOVEL, MOVEC and Linear SPLINE) robot movements in millimeters per second.

Movement commands which do not include a *duration* argument are executed according to the SPEEDL setting.

In DIRECT mode, the SPEEDL command takes effect immediately. Determines the speed of movement when the MOVEL(D), MOVEC(D) and Linear SPLINE commands are executed in DIRECT mode.

In EDIT mode, the SPEEDL command takes effect after it is executed from within a program. Determines the speed of movement when the MOVEL(D), MOVEC(D) and Linear SPLINE commands are executed from within a program.

To view current speed settings, use the SHOW SPEED command.

Examples:

- `SET VARSP 12000`
  `SPEEDL VARSP`
  Sets speed of linear/circular movements of group A to 12 mm/sec.
- `SPEEDL 12.000`
  Sets speed of linear/circular movements of group A to 12 mm/sec.
- `SPEEDL 12`
  Sets speed of linear/circular movements of group A to 12 mm/sec.

Notes: Refer to the MOVE(D), MOVES(D), SPLINE(D), SPEED and SHOW SPEED commands.

WAIT EDIT

Format: `WAIT var1 oper var2`

Where: var1 is a variable;
var2 is a variable or a constant;
oper can be: `<, >, >=, <=, =, <>`

Description: Program execution is suspended until the specified condition is true.

When a program is waiting for an input to reach a specific state, this command is very useful, since WAIT uses little CPU power while waiting for an event.

Examples:

- `WAIT IN[5]=1` Waits until input 5 is ON
- `WAIT X<Y` Wait until the value of X is less than the value of Y.