ACL

Advanced Control Language

Versions 1.43, F.44

Reference Guide

for Controller-A

4th Edition

Catalog #100083 Rev.A

ESHED ROBOTEC

Catalog #100083 Rev.A

(January 1995) Fourth Edition

April 1999 Reprinted/PDF version

Information regarding the ATS software, which was previously included in this manual, is now available in a separate reference guide.

Every effort has been made to make this book as complete and accurate as possible. However, no warranty of suitability, purpose, or fitness is made or implied. Eshed Robotec is not liable or responsible to any person or entity for loss or damage in connection with or stemming from the use of the software and/or the information contained in this publication.

Eshed Robotec bears no responsibility for errors which may appear in this publication and retains the right to make changes to the software and manual without prior notice.

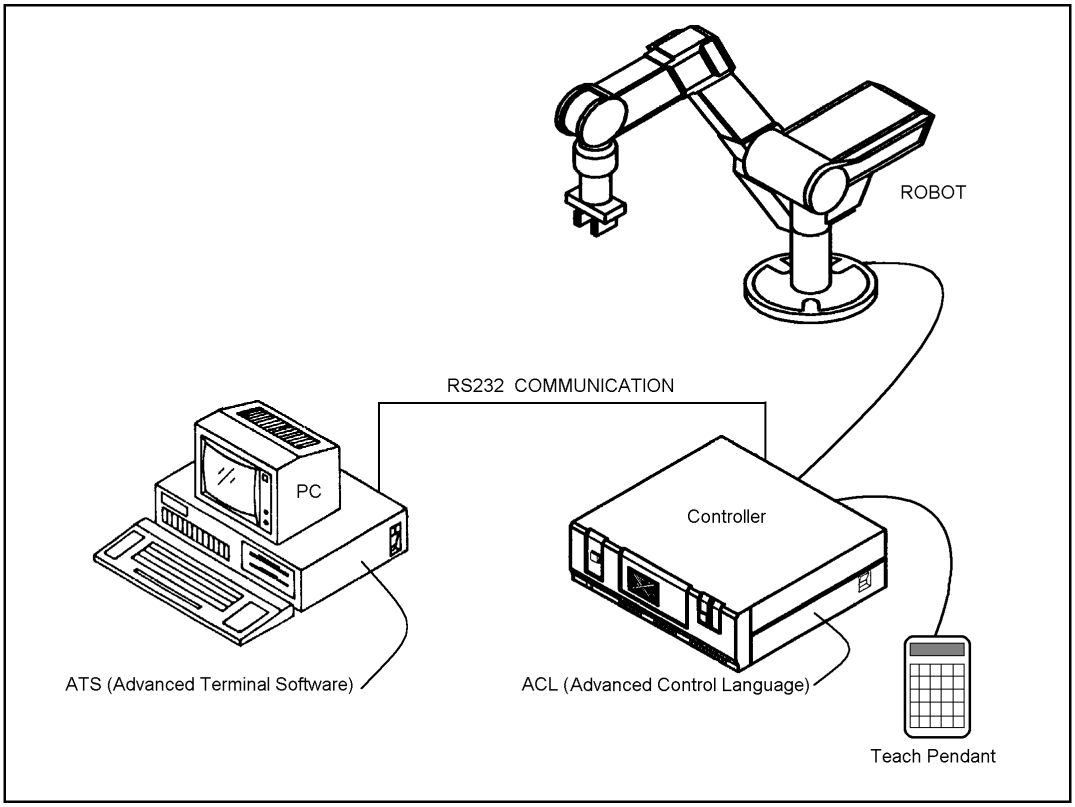
Table of Contents

Introduction

ACL, Advanced Control Language, is an advanced, multi-tasking robotic programming language developed by Eshed Robotec (1982) Ltd. **ACL** is programmed onto a set of EPROMs within **Controller-A**, and can be accessed from any standard terminal or PC computer by means of an RS232 communication channel.

ATS, Advanced Terminal Software, is the user interface to the **ACL** controller. **ATS** is supplied on diskette and operates on any PC host computer. The software is a terminal emulator which enables access to **ACL** from a PC computer.

The following diagram shows the components of the robotic control system.


ROBOT
RS232 COMMUNICATION
PC
Controller
ATS (Advanced Terminal Software)
ACL (Advanced Control Language)
Teach Pendant

ACL features include the following:

- Direct user control of robotic axes.
- User programming of robotic system.
- Input/output data control.
- Simultaneous and synchronized program execution (full multi-tasking support).
- Simple file management.

ATS features include the following:

- Short-form controller configuration.
- Definition of peripheral devices.
- Short-cut keys for command entry.
- Backup manager.
- Print manager.

CHAPTER 1

ACL Programming Language: Quick Reference

This chapter presents a brief summary of the command modes and data types used by **ACL**. These topics are described fully in other chapters of this manual.

In addition, this chapter includes brief descriptions of the **ACL** commands grouped according to the categories listed below. These lists will help you compare and select the command most suitable for your specific programming and operating requirements.

- Axis Control Commands
- I/O Control Commands
- Program Control Commands
- Position Definition and Manipulation Commands
- Variable Definition and Manipulation Commands
- Program Flow Commands
- Configuration Commands
- Report Commands
- User Interface Commands
- Program Manipulation Commands
- Editing Commands
- RS232 Communication Commands
- Backup/Restore Commands

For more detailed descriptions of individual commands, refer to Chapter 3.

Command Modes

ACL has two types of commands:

- DIRECT commands are executed as soon as they are entered at the terminal/computer keyboard.
- EDIT, or indirect, commands are executed during the running of the programs and routines in which they are used.

Some commands can be issued in both the DIRECT and EDIT modes, as indicated throughout this manual.

Refer to Chapter 2 for a detailed explanation of these command modes.

Coordinate Systems

ACL allows robotic systems to be operated and programmed in two different coordinate systems:

- JOINT (encoder) values.
- XYZ (Cartesian) coordinates.

Refer to Chapter 2 for a detailed explanation of the coordinate systems.

Data Types

Variables

ACL uses two types of variables:

- User variables:
  - User defined GLOBAL variables can be used in all programs.
  - User defined PRIVATE variables can only be used in the program which was being edited at the time the variable was defined.
- System variables.

  System defined variables contain values which indicate the status of inputs, outputs, encoders, and other control system elements.

  Refer to Chapter 4 for a detailed explanation of user and system variables.

Strings (Comments)

Some **ACL** command lines include comments or textual strings. Strings of up to 40 characters and spaces are recognized.

Refer to Chapter 2 for a detailed explanation of strings.

Positions

ACL uses six types of positions:

- Absolute Joint
- Absolute XYZ
- Relative to Another Position by Joint
- Relative to Another Position by XYZ
- Relative to Current by Joint
- Relative to Current by XYZ

Refer to Chapter 2 for a detailed explanation of positions.

Parameters

ACL parameters define the values of physical constants which adapt the controller to a particular robotic system.

Parameters are referred by their numbers (1 to 320).

Refer to Chapter 7 for a detailed explanation and description of parameters.

Axis Control Commands

MOVE
MOVED
MOVEC
MOVECD
MOVEL
MOVELD
MOVES
MOVESD

OPEN
CLOSE
JAW

CLRBUF

SPEED
SHOW SPEED

EXACT
MPROFILE

CON
COFF

SET ANOUT
SHOW DAC

INT_ON
INT_OFF
TON
TOFF

HOME
LSON
LSOFF
CLR

~
<Alt>+M

TEST

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
MOVE				
	MOVE *pos*	Moves axes to target position at current joint speed.	DIRECT, EDIT	
	MOVE *pos duration*	Moves axes to target position within time specified.	DIRECT, EDIT	
	MOVED *pos* [*duration*]	Same as MOVE, and suspends program until axes accurately reach target position.	EDIT	Execution is affected by EXACT command.
MOVEC				
	MOVEC *pos1 pos2*	Moves robot’s TCP to position 1, along a circular path through position 2, at current linear speed.	DIRECT, EDIT	
	MOVECD *pos1 pos2*	Same as MOVEC, and suspends program until axes have accurately reached position 2.	EDIT	Execution is affected by EXACT command.
MOVEL				
	MOVEL *pos*	Moves robot’s TCP to target position, along a linear path, at current linear speed.	DIRECT, EDIT	
	MOVEL *pos duration*	Moves robot’s TCP to target position, along a linear path, within time specified.	DIRECT, EDIT	
	MOVELD *pos* [*duration*]	Same as MOVEL, and suspends program until axes accurately reach target position.	EDIT	Execution is affected by EXACT command.

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
MOVES				
	MOVES *pvect pos1 posn*	Moves axes smoothly through all consecutive vector positions between position 1 and position *n*, at current joint speed. Constant speed betweeen consecutive positions.	DIRECT, EDIT	
	MOVES *pvect pos1 posn duration*	Same as MOVES, except average speed determined by time definition.	DIRECT, EDIT	
	MOVESD *pvect pos1 posn*	Same as MOVES, and suspends program until axes accurately reach position *n*.	EDIT	Execution is affected by EXACT command.
OPEN				
	OPEN	Disables servo control of gripper, and opens gripper until end of motion.	DIRECT, EDIT	Standard command for opening gripper.
	OPEN *var*	Disables servo control of gripper; sets gripper DAC to *var*. Opens gripper with additional force.	EDIT	Use with caution. May damage gripper. 0 ≤ *var* ≤ 5000
CLOSE				
	CLOSE	Disables servo control of gripper, and closes gripper until end of motion.	DIRECT, EDIT	Standard command for closing gripper.
	CLOSE *var*	Disables servo control of gripper; sets gripper DAC to *var*. Closes gripper with additional force.	EDIT	Use with caution. May damage gripper. 0 ≤ *var* ≤ 5000
JAW				
	JAW *var*	Enables gripper servo control. Brings gripper jaw to a percentage of fully open. Movement at maximum speed.	DIRECT, EDIT	Use with caution. May damage motor. 0 ≤ *var* ≤100
	JAW *var duration*	Same as JAW, except speed determined by time definition.	DIRECT, EDIT	Use with caution. May damage motor. 0 ≤ *var* ≤100
CLRBUF				
	CLRBUF	Empties movement buffer of all axes.	DIRECT, EDIT	
	CLRBUFA/B	Empties movement buffer of group A or group B.	DIRECT, EDIT	
	CLRBUF *axis*	Empties movement buffer of specific axis.	DIRECT, EDIT	

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
SPEED				
	`SPEED` *var*	Sets speed for all axes.	DIRECT, EDIT	1 ≤ *var*≤ 100. Default is 50.
	`SPEED{A/B}` *var*	Sets speed for group A or B.	DIRECT, EDIT	
	`SPEEDC` *var axis*	Sets speed for axis in group C.	DIRECT, EDIT	
SHOW SPEED				
	`SHOW SPEED`	Displays the current speed settings.	DIRECT	
EXACT				
	`EXACT {A/B/C}`	Ensures movement reaches target position accurately; disregards *duration* if specified in movement command. Defined separately for group A, B and C. Only affects commands with the 'D' suffix: MOVED, MOVELD, MOVECD, MOVESD.	DIRECT, EDIT	EXACT is default mode.
	`EXACT OFF{A/B/C}`	Movement reaches target position according to *duration*; accuracy not guaranteed. Only affects movement commands with the 'D' suffix.	DIRECT, EDIT	
MPROFILE				
	`MPROFILE TRAPEZE {A/B/C}`	Applies trapezoid profile to trajectory: fast acceleration and deceleration at start and end of movement, with constant speed along path.	DIRECT, EDIT	PARABOLE is default mode.
	`MPROFILE PARABOLE {A/B/C}`	Applies paraboloid profile to trajectory: slow acceleration until maximum speed is reached; deceleration at same rate.	DIRECT, EDIT	
CON				
	`CON[A/B]`	Enables servo control of all axes, or specifically of group A or B.	DIRECT	
	`CON` *axis*	Enables servo control of a specific axis.	DIRECT	
COFF				
	`COFF[A/B]`	Disables servo control of all axes, or specifically of group A or B.	DIRECT	
	`COFF` *axis*	Disables servo control of a specific axis.	DIRECT	

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
SET ANOUT				
	SET ANOUT[*n*]=DAC	Disables servo control of a specific axis and sets the DAC value for a specific axis.	DIRECT, EDIT, PRIV	-5000 ≤ DAC ≤ 5000. Use with care. May damage motor.
SHOW DAC				
	SHOW DAC *axis*	Displays the value of DAC in millivolts.	DIRECT	1 ≤ *axis* ≤ 11
INT				
	INT_ON *axis1*...*axis4*	Enables integral servo control of the specified axes.	DIRECT, EDIT	INT_ON is default mode.
	INT_OFF axis1...*axis4*	Disables integral servo control of the specified axes.	DIRECT, EDIT	
TON				
	TON [*n*]	Enables thermic motor protection of all axes, or a specific axis.	DIRECT	TON is default mode.
TOFF				
	TOFF [*n*]	Disables thermic motor protection of all axes, or a specific axis.	DIRECT	*Use with caution.*
HOME				
	HOME [*n*]	Searches for microswitch home position, for all robot axes, or specific axis.	DIRECT, EDIT	From teach pendant, key in: RUN 0. TP homes robot only.
	HHOME *n*	Searches for hard stop home for specific axis.	DIRECT, EDIT	
LSON				
	LSON	Connects axis home switches to the controller’s input sensors. Disables inputs.	DIRECT, EDIT	
LSOFF				
	LSOFF	Disconnects axis home switches from the controller’s input sensors. Enables inputs.	DIRECT, EDIT	LSOFF is default mode.
CLR				
	CLR *n*	Initializes (sets to 0) the value of a specific encoder.	DIRECT, PRIV	1 ≤ *n* ≤ 11
	CLR *	Initializes (sets to 0) the value of all encoders.	DIRECT, PRIV	

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
~				
`~` or `<Ctrl>+M`		Activates and deactivates Manual mode, for direct control of axes from terminal or computer keyboard .	DIRECT	
TEST				
`TEST`		Executes internal diagnostic procedure for testing movement of the axes, and operation of homing microswitches and controller I/Os.	DIRECT	

I/O Control Commands

`DISABLE`
`ENABLE`
`FORCE`
`SHOW DIN`
`SHOW DOUT`
`SET OUT[`*n*`]`
`IF IN[`*n*`]`
`TRIGGER`

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
DISABLE				
	`DISABLE {IN/OUT}` *n*	Disconnects the physical input or output from normal system control.	DIRECT	1 ≤ *n* ≤ 16
	`DISABLE ?`	Displays a list of all disabled inputs and outputs.	DIRECT	
ENABLE				
	`ENABLE {IN/OUT}` *n*	Reconnects a disabled input or output to normal system control.	DIRECT	1 ≤ *n* ≤ 16 ENABLE is default mode.
FORCE				
	`FORCE {IN/OUT}` *n* `{0/1}`	Forces a disabled input or output to a different state.	DIRECT	1 ≤ *n* ≤ 16 0=OFF; 1=ON
	`FORCE ?`	Displays the state of all forced inputs and outputs.	DIRECT	Display: 1=ON; 0=OFF
SHOW				
	`SHOW DIN`	Displays the state of all 16 inputs.	DIRECT	Display: 1=ON; 0=OFF
	`SHOW DOUT`	Displays state of all 16 outputs.	DIRECT	Display: 1=ON; 0=OFF
SET				
	`SET OUT[`*n*`]={0/1}`	Sets the state of an output port.	DIRECT, EDIT	1 ≤ *n* ≤ 16 0=OFF; 1=ON
IF				
	`IF IN[`*n*`]={0/1}`	Checks the state of an input.	EDIT	1 ≤ *n* ≤ 16 0=OFF; 1=ON
TRIGGER				
	`TRIGGER` *prog* `BY {IN/OUT}` *n* `{0/1}`	Executes a program, conditional upon a change in the state of an input or output.	EDIT	1 ≤ *n* ≤ 16 0=OFF; 1=ON

Program Control Commands

RUN
A
STOP
SUSPEND
CONTINUE

PRIORITY

SET *var* TIME

DELAY
WAIT
TRIGGER BY IN/OUT

PEND
POST
QPEND
QPOST

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
RUN				
	`RUN` *prog*	Runs the specified program.	DIRECT, EDIT	
	`RUN` *prog priority*	Runs the specfied program, subject to priority.	DIRECT, EDIT	
A				
	`A` or `<Ctrl>+A`	Immediately aborts all running programs, and stops axes movement.	DIRECT	
	`A` *prog*	Aborts the specified program.	DIRECT	
STOP				
	`STOP`	Aborts all running programs.	EDIT	
	`STOP` *prog*	Aborts a specific running program.	EDIT	
SUSPEND				
	`SUSPEND` *prog*	Halts execution of a program.	DIRECT, EDIT	
CONTINUE				
	`CONTINUE` *prog*	Resumes execution of aprogram previously halted by SUSPEND.	DIRECT, EDIT	
PRIORITY				
	`PRIORITY` *prog var*	Sets a program's run time priority to *var*. Programs with a higher priority have precedence when the CPU is loaded.	EDIT	1 ≤ *var* ≤ 10. Default is 5.
SET				
	`SET` *var*`=TIME`	Assigns the value of system variable TIME to *var*.	DIRECT, EDIT	

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
DELAY				
	DELAY *var*	Suspends program execution for the time specified by *var*.	EDIT	*var* defined in hundredths of a second.
WAIT				
	WAIT *var1 oper var2*	Suspends program execution until condition is satisfied (true).	EDIT	*Cond* can be*:* `<,>,=,<=,>=,<>`
TRIGGER				
	TRIGGER *prog* BY {IN/OUT} *n* {0/1}	Executes a program, conditional upon a change in the state of an input or output.	EDIT	1 ≤ *n* ≤ 16 0=OFF; 1=ON
PEND				
	PEND *var1* FROM *var2*	Suspends program execution until another program posts a non-zero value to *var2*.	EDIT	Used with POST to synchronize programs.
POST				
	POST *var3 TO var2*	Assigns the value of *var3* to *var2*.	EDIT	Used with PEND to synchronize programs.
QPEND				
	QPEND *var1* FROM *array*	Same as PEND, but value is taken from a queue (a variable array).	EDIT	Used with QPOST to synchronize programs.
QPOST				
	QPOST *var*3 TO *array*	Same as POST but value is put into a queue (a variable array).	EDIT	Used with QPEND to synchronize programs.

Position Definition and Manipulation Commands

DEFP
DIMP

DELP
UNDEF

HERE
HERER

TEACH
TEACHR

SETPV
SETPVC

SHIFT
SHIFTC

SETP

ATTACH

SET *var*=PVAL
SET *var*=PVALC
SET *var*=PSTATUS

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
DEFP				
	`DEFP[A/B]` *pos* `DEFPC` *pos axis*	Defines (creates) a position for group A or B or for axis in group C.	DIRECT, EDIT	1 ≤ *axis* ≤ 11
DIMP				
	`DIMP[A/B]` *vect*[*n*] `DIMPC` *vect*[*n*] *axis*	Defines (creates) a vector of *n* positions for group A or B or for axis in C.	DIRECT, EDIT	1 ≤ *axis* ≤ 11
DELP				
	`DELP` *pos* `DELP` *pvect*	Deletes positions and position vectors from user RAM.	DIRECT, EDIT	
UNDEF				
	`UNDEF` *pos*	Deletes position coordinate values, but *position is still defined*.	DIRECT, EDIT	
	`UNDEF` *pvect*	Deletes coordinate values of all positions in the vector, but *vector is still defined*.	DIRECT, EDIT	
HERE				
	`HERE` *pos*	Records joint coordinates for current position of axes.	DIRECT, EDIT	Joint coordinates = encoder counts.
HERER				
	`HERER` *pos*	Records joint offset coordinates for a position relative to the current position.	DIRECT	
	`HERER` *pos2 pos1*	Records joint offset coordinates for a position relative to another position.	DIRECT, EDIT	

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
TEACH				
	`TEACH` *`pos`*	Records Cartesian coordinates for a robot position.	DIRECT	
TEACHR				
	`TEACHR` *`pos`*	Records Cartesian offset coordinates for a robot position relative to the current robot position.	DIRECT	
	`TEACHR` *`pos2 pos1`*	Records Cartesian offset coordinates for a robot position relative to another robot position.	DIRECT	
SETPV				
	`SETPV` *`pos`*	Records joint coordinates for a position.	DIRECT	
	`SETPV` *`pos axis var`*	Changes one joint coordinate of a previously recorded position.	DIRECT, EDIT	1 ≤ *axis* ≤ 11
SETPVC				
	`SETPVC` *`pos coord var`*	Changes one Cartesian coordinate of a previously recorded robot position.	DIRECT, EDIT	*coord*={X/Y/Z/P/R}
SHIFT				
	`SHIFT` *`pos`* `BY` *`axis var`*	Changes one joint coordinate of a previously recorded position *by an offset value*.	DIRECT, EDIT	1 ≤ *axis* ≤ 11
	`SHIFTC` *`pos`* `BY` *`coord var`*	Changes one Cartesian coordinate of a previously recorded robot position *by an offset value*.	DIRECT, EDIT	*coord*={X/Y/Z/P/R}
SETP				
	`SETP` *`pos2=pos1`*	Copies the coordinates and type of *pos1* to *pos2*.	DIRECT, EDIT	
ATTACH				
	`ATTACH` *`pvect`*	Attaches a position vector to the teach pendant according to group for which the vector is defined. Vector positions can now be accessed from TP by means of their index numbers.	DIRECT	
	`ATTACH OFF{A/B/C}`	Detaches the position vector which is currently attached to the TP. Group A, B or C must be specified.	DIRECT	
	`ATTACH ?`	Displays current ATTACH status.	DIRECT	

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
SET				
	`SET` *`var`*`=PVAL` *`pos axis`*	Assigns *var* the value of one joint coordinate of a recorded position.	DIRECT, EDIT	1 ≤ *axis* ≤ 11
	`SET` *`var`*`=PVALC` *`pos coord`*	Assigns *var* the value of one Cartesian coordinate of a recorded position.	DIRECT, EDIT	*coord*={X/Y/Z/P/R}
	`SET` *`var`*`=PSTATUS` *`pos`*	Assigns *var* a value according to the type of the position.	DIRECT, EDIT	

Variable Definition and Manipulation Commands

DEFINE
GLOBAL

DIM
DIMG

DELVAR

SET *var*

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
DEFINE				
	`DEFINE` *`var1...var12`*	Creates (defines) private variables. Up to 12 variables can be defined in one command.	EDIT	A private variable is recognized only by the program in which it is defined.
GLOBAL				
	`GLOBAL` *`var1...var12`*	Creates (defines) a global variable. Up to 12 variables can be defined in one command	DIRECT, EDIT	Global variables can be used by any programs.
DIM				
	`DIM` *`var`*`[`*`n`*`]`	Creates (defines) an array of *n* private variables.	EDIT	
DIMG				
	`DIMG` *`var`*`[`*`n`*`]`	Creates (defines) an array of *n* global variables.	DIRECT, EDIT	
DELVAR				
	`DELVAR` *`var`*	Deletes variable from user RAM.	DIRECT, EDIT	
SET (arithmetic and logical functions)				
	`SET` *`var1`*`=`*`var2`*	Assigns the value of *var2* to *var1*.	DIRECT, EDIT	
	`SET` *`var1`* `NOT` *`var2`*	Assigns the logical negative value of *var2* to *var1*.	DIRECT, EDIT	
	`SET` *`var1`*`=COMPLEMENT` *`var2`*	Assigns the complement value of *var2* to *var1*.	DIRECT, EDIT	
	`SET` *`var1`*`= ABS` *`var2`*	Assigns the absolute value of *var2* to *var1*.	DIRECT, EDIT	
	`SET` *`var1`*`=` *`var2 oper var3`*	Assigns to *var1* the result of the operation on the other two variables.	DIRECT, EDIT	*oper*: +, –, *, /, SIN, COS, TAN, ATAN, EXP, LOG, MOD, OR, AND

Program Flow Commands

`IF`	`FOR`	`LABEL`
`ANDIF`	`ENDFOR`	`GOTO`
`ORIF`		
`ELSE`		`GOSUB`
`ENDIF`		

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
IF	`IF` *`var1 oper var2`*	Checks the conditional relation of two variables.	EDIT	*oper* can be: `<,>,=,<=,>=,<>`
ANDIF	`ANDIF` *`var1 oper var2`*	Logically combines a condition with other IF commands.	EDIT	
ORIF	`ORIF` *`var1 oper var2`*	Logically combines a condition with other IF commands.	EDIT	
ELSE	`ELSE`	Follows IF and precedes ENDIF. Begins subroutine when IF is false.	EDIT	
ENDIF	`ENDIF`	End of IF subroutine.	EDIT	
FOR	`FOR` *`var1=var2`* `TO` *`var3`*	Loop command. Executes subroutine for all values of variable.	EDIT	
ENDFOR	`ENDFOR`	End of FOR loop.	EDIT	
LABEL	`LABEL` *`n`*	Marks a program subroutine to be executed by GOTO command.	EDIT	$0 \leq n \leq 9999$
GOTO	`GOTO` *`label_n`*	Continues program execution at line following specified LABEL.	EDIT	
GOSUB	`GOSUB` *`prog`*	Transfers control to another program. Main program is suspended until subroutine is completed.	EDIT	

Configuration Commands

CONFIG

LET PAR
SHOW PAR

INIT EDITOR
INIT CONTROL
INIT PROFILE

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
CONFIG				
	`CONFIG`	Activates procedure for configuring the controller configuration.	DIRECT	Automatically followed by INIT EDITOR; Erases user RAM.
	`CONFIG ?`	Displays the current controller configuration.	DIRECT	
LET PAR				
	`LET PAR` *n*=*var*	Changes the value of system parameters.	DIRECT PRIV	Must be followed by INIT CONTROL. *Use with caution !*
SHOW				
	`SHOW PAR` *n*	Displays the value of parameter *n*.	DIRECT	
INIT				
	`INIT EDITOR`	Erases all user programs, positions and variables in user RAM.	DIRECT	*Use with caution !*
	`INIT CONTROL`	Resets system parameters according to LET PAR values.	DIRECT	Must be executed after changing parameters.
	`INIT PROFILE`	Initializes the velocity profiles according to the value of parameter 76.	DIRECT	Must be executed after changing parameter 76.

Report Commands

```
ATTACH ?          SHOW              DIR
CONFIG ?          STAT              LIST
DISABLE ?         VER               SEND
FORCE ?           FREE
```

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
ATTACH				
	`ATTACH ?`	Displays current ATTACH status.	DIRECT	
CONFIG				
	`CONFIG ?`	Displays the current controller configuration.	DIRECT	
DISABLE				
	`DISABLE ?`	Displays a list of all disabled inputs and outputs.	DIRECT	
FORCE				
	`FORCE ?`	Displays a list of all forced inputs and outputs.	DIRECT	
SHOW				
	`SHOW DIN`	Displays status of all 16 inputs. If LSON mode is active, will display status of all homing microswitches.	DIRECT	Display: 1=Input ON 0=Input OFF
	`SHOW DOUT`	Displays status of all 16 outputs.	DIRECT	Display: 1=Output ON 0=Output OFF
	`SHOW ENCO`	Displays the values of all encoders every 0.5 seconds	DIRECT	<Ctrl>+C cancels the display.
	`SHOW DAC` *`axis`*	Displays the value of DAC in millivolts.	DIRECT	$1 \leq axis \leq 11$
	`SHOW PAR` *`n`*	Displays the value of parameter *n*.	DIRECT	
	`SHOW SPEED`	Displays the current speed settings.	DIRECT	
STAT				
	`STAT`	Displays a list of active user programs: name, priority, status.	DIRECT	
VER				
	`VER`	Displays ACL EPROM version.	DIRECT	
FREE				
	`FREE`	Displays a list of available user memory.	DIRECT	

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
DIR				
	`DIR`	Displays a list of the names and ID numbers of all user programs.	DIRECT	
LIST				
	`LIST [`*prog*`]`	Displays all lines of all user programs or a specific program.	DIRECT	
	`LISTP`	Displays a list of all defined positions.	DIRECT	
	`LISTPV` *pos*	Displays the type of position and joint coordinates of the specified position. Cartesian coordinates also displayed for robot positions.	DIRECT	
	`LISTPV POSITION`	Displays current coordinates of robot arm.	DIRECT	
	`LISTVAR`	Displays a list of all user and system variables.	DIRECT	
SEND				
	`SEND`	Displays all user programs, variables and positions, and parameters in RECEIVE/APPEND format.	DIRECT	
	`SEND` *prog*	Displays the specified user program in RECEIVE *prog* format.	DIRECT	
	`SENDPROG`	Displays all user programs, variables, and positions in RECEIVE/APPEND format.	DIRECT	
	`SENDPOINT`	Displays all user defined positions in RECEIVE/APPEND format.	DIRECT	
	`SENDVAR`	Displays all user defined variables in RECEIVE/APPEND format.	DIRECT	
	`SENDPAR`	Displays all system parameters in RECEIVE/APPEND format.	DIRECT	

User Interface Commands

`QUIET`	`HELP`	`READ`
`NOQUIET`		`GET`
	`PRINT`	
`ECHO`	`PRINTLN`	`DO`
`NOECHO`		

FORMAT	DESCRIPTION	MODE	NOTES
QUIET			
`QUIET`	DIRECT commands in running program are not displayed on screen.	DIRECT	
NOQUIET			
`NOQUIET`	DIRECT commands in running program are displayed on screen.	DIRECT	NOQUIET is default mode.
ECHO			
`ECHO`	Displays on screen all characters that are transmitted to controller.	DIRECT	ECHO is default mode.
NOECHO			
`NOECHO`	Keyboard entries are not displayed on screen.	DIRECT	
HELP			
`HELP`	Provides on-line help for EDIT commands.	EDIT	
`HELP`	Provides on-line help for DIRECT commands.	DIRECT	
`DO HELP`	Provides on-line help for EDIT commands.	DIRECT	
PRINT			
`PRINT "`*`string`*`"`	Displays *string* on screen.	DIRECT, EDIT	
`PRINT `*`var1...var4`*	Displays the value(s) of specified variable(s).	DIRECT, EDIT	
PRINTLN			
`PRINTLN`	Same as PRINT, but starts a new line before displaying text.	EDIT	

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
READ				
	`READ "`*`string`*`" `*`var`*	Displays the *string* and waits for value of *var* from keyboard.	EDIT	
GET				
	`GET `*`var`*	Waits for one keyboard character to be pressed. ASCII value of character is assigned to *var*.	EDIT	
DO				
	`DO `*`editcom`*	Executes certain EDIT mode commands when controller in DIRECT mode.	DIRECT	

Program Manipulation Commands

COPY
RENAME
REMOVE
EDIT

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
COPY				
	`COPY` *prog1 prog2*	Copies program *prog1* to a new program *prog2*	DIRECT	
RENAME				
	`RENAME` *prog1 prog2*	Changes name of user program from *prog1* to *prog*2.	DIRECT	
REMOVE				
	`REMOVE` *prog*	Deletes program from user RAM.	DIRECT	
EDIT				
	`EDIT` *prog*	Activates EDIT mode for program creation and editing.	DIRECT	

Editing Commands

S
P
L
DEL
<Enter>
*
@
EXIT
END

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
S				
	`S`	Goes to the first line of the program being edited.	EDIT	
	`S` *`n`*	Goes to line *n* of program being edited.	EDIT	
P				
	`P`	Goes to previous line of program.	EDIT	
L				
	`L` *`n1 n2`*	Displays program lines, from line *n1* to line *n2*.	EDIT	
DEL				
	`DEL`	Erases the current line of program.	EDIT	
<Enter>				
	`<Enter>`	Goes to next line in program and displays its number.	EDIT	
*				
	`*` *`string`*	* precedes user comment line.	EDIT	
@				
	`@` *`DIRECTcom`*	Allows the execution of a DIRECT command from a running user program.	EDIT	
EXIT				
	`EXIT`	Quits EDIT and checks program validity.	EDIT	
END				
	`END`	End of program. Automatically written by system at end of program	EDIT	Not a user command.

RS232 Communication Commands

SENCOM	PRCOM	CLRCOM
GETCOM	PRLNCOM	
	READCOM	

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
SENCOM				
`SENCOM` *`n var`*		Transmits one byte, whose value is specified by a variable or constant, to the specified RS232 port.	EDIT	*n*=RS232 COM port; 1≤ *n* ≤ 8
GETCOM				
`GETCOM` *`n var`*		Receives one byte from the specified RS232 port, and stores its value in the specified variable.	EDIT	1 ≤ *n* ≤ 8
PRCOM				
`PRCOM` *`n arg1`* `[`*`arg2 arg3`*`]`		Transmits *arg* (strings and/or variable values) to the specified RS232 port.	EDIT	1 ≤ *n* ≤ 8
PRLNCOM				
`PRLNCOM` *`n arg1`* `[`*`arg2 arg3`*`]`		Transmits *arg* (strings and/or variable values) to the specified RS232 port, and adds carriage return.	EDIT	1 ≤ *n* ≤ 8
READCOM				
`READCOM` *`n var`*		Receives ASCII character(s) followed by a carriage return (↵) from the specified RS232 port and assigns the ASCII numeric value to *var*.	EDIT	1 ≤ *n* ≤ 8
CLRCOM				
`CLRCOM` *`n`*		Clears the buffer of the specified RS232 port, or all ports.	EDIT	1 ≤ *n* ≤ 8; 0 = all ports

Backup/Restore Commands

```
SEND
RECEIVE
APPEND
```

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
SEND				
`SEND`		Generates a listing of all user programs, variables and positions, and parameters in a format compatible with the RECEIVE and APPEND commands.	DIRECT	
`SEND` *`prog`*		Generates a listing of the specified user program in a format compatible with the RECEIVE *prog* command.	DIRECT	
`SENDPROG`		Generates a listing of all user programs, variables, and positions in a format compatible with the RECEIVE and APPEND commands.	DIRECT	
`SENDVAR`		Generates a listing of all user defined variables in a format compatible with the RECEIVE and APPEND commands.	DIRECT	
`SENDPOINT`		Generates a listing of all user defined positions in a format compatible with the RECEIVE and APPEND commands..	DIRECT	
`SENDPAR`		Generates a listing of all system parameters in a format compatible with the RECEIVE and APPEND commands.	DIRECT	
RECEIVE				
`RECEIVE`		Loads programs, positions and variables from external backup file to user RAM.	DIRECT	Erases current contents of user RAM
`RECEIVE` *`prog`*		Loads contents of one program from backup file.	DIRECT	Does not affect other data in user RAM.
APPEND				
`APPEND`		Adds user programs from external file to user RAM.	DIRECT	Does not affect other data in user RAM.

CHAPTER 2

Command Modes and Formats

This chapter describes the various modes of **ACL** programming and operation, as well as the types and formats of commands and data used in the **ACL** programming language.

Command Modes

Once **ATS** has been loaded, you can communicate with the controller from your computer keyboard. You may now create or edit your programs, or assume direct control of the robot and peripheral axes, depending on the active mode of operation.

ACL has two types of commands:

- **DIRECT** commands, which are executed as soon as they are entered at the terminal/computer keyboard.
- Indirect, or **EDIT** commands, which are executed during the running of the programs and routines in which they are used.

Some commands are available in both the DIRECT mode and the EDIT mode.

DIRECT Mode

When DIRECT mode is active, all commands entered from the keyboard are immediately executed by the controller.

Whenever the DIRECT mode is active, the screen shows the following cursor prompt:

```
>_
```

DIRECT mode commands can be included in programs for execution from a running program by prefacing them with the character @. The @ signals to the controller that the string be read as a DIRECT mode command, and activated from a running program.

Once the @ command has been transmitted, and its execution has begun, the program continues running regardless of the @ command's status. Use the DELAY command to ensure completion of a @ command.

EDIT mode commands can be executed in the DIRECT mode when preceded by the command DO.

Refer to the command descriptions for @, DELAY, and DO in Chapter 3.

Manual Keyboard Control

When in DIRECT mode, you can assume direct control of the robot and peripheral axes from the keyboard by activating Manual mode. This mode is useful when a teach pendant is not available.

To activate the Manual mode, type either of the following:

`<Alt>+m`	(when using **ATS**)
`~`	(usually by pressing <Shift>+‘)

The commands which can be executed in Manual mode are comparable to those available from the teach pendant.

Refer to the command ~ (Manual Keyboard Mode) in Chapter 3 for a complete description of the functions available in Manual mode.

Teach Pendant Control

The teach pendant is a hand-held terminal which permits the operator direct control of the robot and peripheral axes. In addition to controlling movement of the axes, the teach pendant is used for recording positions, sending axes to recorded positions, activating programs, and other functions.

The teach pendant provides direct control of the axes even when the controller is in EDIT mode.

Teach pendant operation is described fully in the *User's Manual* supplied with your robot/controller.

EDIT Mode

The EDIT mode is used to create and edit **ACL** programs.

Whenever the EDIT mode is active, the screen shows the current program line number and a cursor prompt, indicating that a command can be inserted. For example:

```
143:?_
```

The controller assigns the line numbers; they are not user definable.

The EDIT mode is activated by typing the command EDIT and the name of a program. For example:

```
>edit pack1
```

The system will respond:

```
PACK1 NEW PROGRAM
DO YOU WANT TO CREATE THAT PROGRAM (Y/N)>
```

Type:

```
y <Enter>
```

The system will respond:

```
         PROGRAM    PACK1
         ********************
36:?_
```

If you do not specify the name of a program after the EDIT command, you will be prompted to provide one.

If you have specified the name of an existing program after the EDIT command, you will be prompted as follows:

```
WELCOME TO ACL EDITOR, TYPE HELP WHEN IN TROUBLE.
         PROGRAM    PACK1
         ********************
36:?_
```

The cursor is located at the first line of program PACK1.

Names used to define programs may be a combination of up to five alphanumeric characters. For example:

`RUN` **`MILL3`**	Executes the program MILL3.
`GOSUB` **`20`**	Execution goes to the first line of the program named 20.

Editing Functions

ACL provides the following EDIT mode commands for program editing:

`S`	Goes to the first line of the program.
`P`	Goes to the preceding line.
`L` *`n1 n2`*	Displays program lines, from the first line specified, to the last line specified.
`DEL`	Erases the current line of the program.
`<Enter>`	Goes to the next line in the program and displays the line number and a cursor prompt (EDIT mode). Or, checks and inserts the currently typed command line (DIRECT mode).
`EXIT`	Quits EDIT mode, and checks program validity.

Refer to the complete descriptions for each of these commands in Chapter 3.

ATS utilizes the following keys for editing commands. Note that these keys can be used in both EDIT and DIRECT mode.

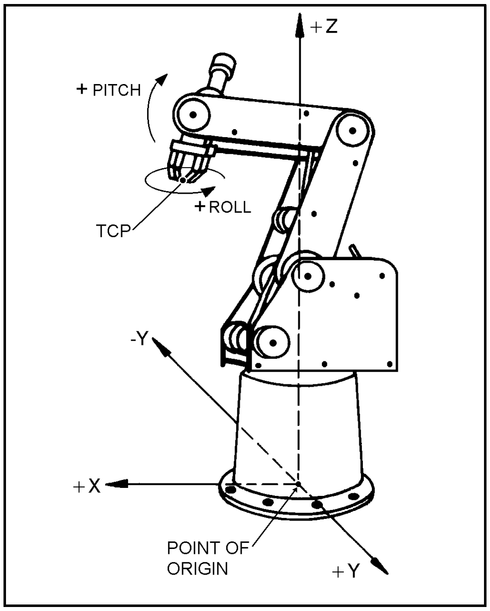
←	(or backspace) Removes characters.
→	Restores characters.
`<Ins>`	Inserts characters.
`<Del>`	Erases characters.
`<Esc>`	Erases the currently typed command.
`<Ctrl>+`→	Restores the currently erased command.
↑	Repeats the last command(s) entered.

Coordinate Systems

ACL allows robotic systems to be operated and programmed in two different coordinate systems: **Joint** coordinates and **Cartesian (XYZ)** coordinates.

Refer to the command ~ (Manual mode) in Chapter 3 for a complete description of axes movements in each of these modes.

Cartesian (XYZ) Coordinates



The Cartesian, or XYZ, coordinate system is a geometric system used to specify the position of the gripper tip by defining its distance, in linear units, from the point of origin (the center bottom of the robot base), along three linear axes, as shown in the illustration here.

To complete the position definition, the pitch and roll of the gripper are specified in angular units.

When robot motion is executed in XYZ mode, all or some of the axes move in order to move the tip of the gripper along an X, Y and/or Z axis.

Joint Coordinates

Joint coordinates specify the location of each axis in encoder counts. When the axes move, their optical encoders generate a series of alternating high and low electrical signals. The number of pulses generated is proportional to the amount of axis motion. The controller counts the pulses and determines how far an axis has moved. Similarly, a robot movement or position can be defined as a specific number of encoder counts for each axis, relative to the home position or to another coordinate.

When robot motion is executed in JOINT mode, individual axes move according to the command.

The position of any peripheral devices which are connected to the system is always according to encoder counts.

Data Types

The **ACL** programming language uses four data types:

- Variables
- Strings
- Positions
- Parameters

Variables

Variables are reserved memory locations which hold integer values in the range: –2147483647 to +2147483647 (long integer, 32 bits).

ACL uses two types of variables: user variables and system variables.

User Variables

User variables may be either global or private.

- **Global Variables**

  Global variables can be used in all programs.

  The command GLOBAL is used to define a global variable.

  The command DIMG is used to define an array of global variables.

- **Private Variables**

  Private variable can only be used in the program which was being edited at the time DEFINE *var*[*n*] or DIM *var*[*n*] was issued.

  The command DEFINE is used to define a private variable.

  The command DIM is used to define an array of private variables.

Up to twelve variables can be defined in one command.

Names used to define variables may be a combination of up to five alpha numeric characters. The first character of a variable name must be a letter. Names of variable arrays also include an index (a number within square brackets) which defines the number of variables in the array.

The following are examples of commands with variables:

`DEFINE` **`X`**	Defines private variable X.
`GLOBAL` **`VAR99`**	Defines global variable VAR99.
`DIM` **`A[20]`**	Defines an array named A containing 20 private variables.
`SET` **`Z`**`=10`	Variable Z is assigned a value of 10.

`SET OUT[3]=` **`Y`** The state of output 3 is determined by the value of variable Y.

`SET` **`Y`**`=IN[1]` The value of variable Y is determined by the status of input 1.

`WAIT IN[`**`J`**`]=1` Condition for variable input J.

User variables have a read/write attribute. You can perform operations on these variables and change their values, using all available **ACL** commands.

The maximum number of user variables is defined by the controller configuration.

System Variables

System defined variables contain values which indicate the status of inputs, outputs, encoders, and other control system elements. The **ACL** system variables enable you to perform diagnostic tests and recovery programs, and to execute applications which require real-time information about the system status.

System variables can be used in the same manner as user variables. However, system variables cannot be deleted, and most system variables are read-only.

ACL for **Controller-A** contains 9 system variables:

```
IN[16]           TIME             MFLAG
OUT[16]          LTA              ERROR
ENC[11]          LTB              ANOUT[11]
```

The indices indicate the dimensions of the array variables.

The values of system variables IN, ENC, TIME, LTA, LTB are updated at each controller clock tick; MFLAG is updated continuously during axis movement. Since any value written to these variables will be overwritten immediately, they are considered read-only variables.

The variables OUT and ANOUT are read/write variables. The values of these system variables are applied at each controller clock tick.

Refer to Chapter 4 for a complete description of system variables.

Variable Lists

The command LISTVAR displays a list of all system and user variables. The name of the program to which a private variable is dedicated appears in parentheses next to the variable.

The command SENDVAR produces a coded list for downloading the variable. The code format is as follows:

Prefix: type of variable ($l for private; $v for global)
Sequential number
Name of variable
Name of dedicated program (for private variable)
Initial value

Strings (Comments)

A string (comment) is an argument of up to 10 characters used in the following ACL commands:

```
PRINT "...."

PRINTLN "...."

PRCOM "...."

PRLNCOM "...."

@ command

* comment
```

Up to 40 characters and spaces—that is, four strings— may comprise the text on these command lines.

If a string is longer than 10 characters, it is automatically divided into substrings, each of which is limited to 10 characters. For example:

```
PRINT "HELLO, HOW DO YOU FEEL THIS MORNING?"
```

This string is actually four arguments:

```
"HELLO, HOW" " DO YOU FE" "EL THIS MO" "RNING?"
```

The maximum number of strings (comments) is defined by the controller configuration.

Positions

Positions are reserved memory locations which hold position data. The position data include one integer value for each axis in the range –32768 to +32767 to define the coordinates, and one word in the range –32768 to +32767 to indicate the type of position.

Types of Positions

ACL has six types of positions, as listed below. The commands used to record each type of position appears in parentheses.

- **Absolute Joint** (HERE, SETPV, SHIFT)

  Position data are the coordinates of the position in encoders values.

- **Absolute XYZ** (TEACH, SETPVC, SHIFTC)

  Position data are the coordinates of the position in Cartesian coordinate values.

- **Relative to Another Position by Joint** (HERER *pos2 pos1*)

  Position data are the differences between encoder values at one position and encoder values at another position.

  ACL permits relative positions to be linked to one another in a chain of up to 32 positions. This relative chain of positions must be anchored to one absolute (root) position.

- **Relative to Another Position by XYZ** (TEACHR *pos2 pos1*)

  Position data are the differences between the Cartesian coordinate values at one position and the Cartesian coordinate values at another position.

  ACL permits relative positions to be linked to one another in a chain of up to 32 positions. This relative chain of positions must be anchored to one absolute (root) position.

- **Relative to Current by Joint** (HERER *pos*)

  Position data are calculated by adding the encoder values at one position to the encoder values at the current position.

  The current position is the encoder values at time the command using the position is executed.

- **Relative to Current by XYZ** (TEACHR *pos*)

  Position data are calculated by adding the Cartesian coordinate values at one position to the Cartesian coordinate values at the current position.

  The current position is the Cartesian coordinate values at time the command using the position is executed.

Defining Positions

The commands DEFP, DEFPB, DEFPC are used to define positions, and the commands DIMPA, DIMPB and DIMPC are used to define position vectors.

To define a position is to reserve a location in controller memory and give a name to the location.

Two types of position names are possible:

- Numerical names (such as 3, 22, 101) of up to five digits. These positions can be accessed directly from the teach pendant.
- Alphanumeric names (such as P, POS10, A2). The name may be a combination of up to five characters, and should begin with a letter. Non-vector positions with alphanumeric names cannot be accessed from the teach pendant.

  Position vectors must have alphanumeric names, which must begin with a letter. The definition also includes an index (a number within square brackets) which defines the number of positions in the vector.

  Positions belonging to vectors can be accessed from the teach pendant when the vector is "attached" to the teach pendant by means of the ATTACH command. The position can thus be accessed through use of its index number.

Position memory is allocated separately to each of the three axis control groups: group A, group B and group C (individual axes). The maximum number of positions for each group is defined by the controller configuration.

Once a position has been defined, it remains dedicated to a specific axis control group, and cannot accept coordinate values for another axis group. By default, positions are defined for group A.

The following are examples of position definition commands:

`DEFP PA`	Defines one position named PA for group A.
`DIMP AA[10]`	Defines a vector of 10 positions named AA for group A.
`DEFPB PB`	Defines one position named PB for group B.
`DEFPC PC 8`	Defines one position named PC for group C axis 8.
`DIMPC AC[10] 8`	Defines a vector of 10 positions named AC for group C axis 8.

Recording Positions

The commands HERE, HERER, TEACH, TEACHR, SETPV, and SETPVC are used to record coordinate values of defined positions.

To record a position is to write its values in the reserved memory

To record a position is to write its values in the reserved memory location.

The following chart summarizes the commands for position recording.

Records Position	**for All Axis Groups in Joint Coordinates**		**for Robot (group A) only in Cartesian Coordinates**	
Absolute; current values.	`HERE` *`pos`*	DIRECT EDIT	*no command*	
Absolute; user defined values.	`SETPV` *`pos`*	DIRECT	`TEACH pos`	DIRECT
Relative to Current Position.	`HERER` *`pos`*	DIRECT	`TEACHR pos`	DIRECT
Relative to Another Position.	`HERER` *`pos2 pos1`*	DIRECT EDIT	`TEACHR` *`pos2 pos1`*	DIRECT
Absolute; user changes value of recorded position.	`SETPV` *`pos axis var`*	DIRECT EDIT	`SETPVC` *`pos coord var`*	DIRECT EDIT
Absolute; user changes value of recorded position by offset value.	`SHIFT` *`pos`* `BY` *`axis var`*	DIRECT EDIT	`SHIFTC` *`pos`* `BY` *`coord var`*	DIRECT EDIT

Although positions values are recorded in either the Joint or Cartesian coordinate system, the axes can be instructed to move to positions in either coordinate system. The controller converts the coordinate values according to the movement command which is issued.

If a position is defined but not recorded, attempts to execute commands which refer to that position will cause run time errors.

It is recommended that you define (but not necessarily record) positions before editing the program in which they are used.

The following are examples of position recording commands:

`HERE 1` Records the current coordinates of the axes in encoder values, for position 1.

`TEACH P1` Records Cartesian coordinates for position P1.

Position Lists

The command LISTP displays a list of all positions and the group to which each position is dedicated.

The command LISTPV displays the encoder and/or Cartesian coordinate values of a specified position.

The command SENDPOINT produces a coded list for downloading the position. The code format is as follows:

Prefix ($p)
Sequential number
Group (1/2/3: respectively, group A, B, C)
Name of position
Coordinates values
Axis number (if group C)
Type of position

Parameters

Parameters are reserved memory locations which are used to set the values of physical constants needed to adapt the controller to a particular robotic system.

Parameters are referred by their number (1 to 320). For example:

`SHOW PAR 300`	Displays value of parameter 300.
`LET PAR 294 8000`	Sets value of parameter 294 to 8000.

Refer to Chapter 7 for a complete description of system parameters.

Notational Conventions Used in this Manual

The following notations are used in the command formats described and explained throughout this manual:

`{ }` Curly braces enclose a list from which you must choose an item.

`[ ]` Square brackets encloses optional items.
Note, however, that the **ACL** format requires square brackets around the indices of position vectors, variable arrays and inputs/outputs.

`. . .` An ellipsis indicates you may repeat the preceding item zero or more times.

`/` A slash separates alternative items in a list. For example,
`ATTACH OFF{A/B/C}` means:

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

italics Italics represents a descriptive item that should be replaced with an actual item name or value. The most common items are as follows:

Program: *prog*
Variables: *var*
Value: *value*
Axis: *axis*

Position: *pos*
Position vector: *pvect*
Duration (time): *duration*
Argument: *arg*

`>`**`bold`**
`:?`**`bold`** In some examples, bold text is used to indicate command entry; often followed by non-bolded text indicating the controller's response.

Additional Notes

- **ACL** is not case-sensitive. Characters may be entered in either lower case or upper case.

- <Enter>must be pressed following all but three **ACL** commands, and is therefore not usually shown in this manual.

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

`<Ctrl>+A`	Abort.
`~`	Toggles Keyboard Manual mode on and off.
`<Ctrl>+C`	Cancels commands, such as LIST, SHOW ENCO and SEND.

CHAPTER **3**

The ACL Commands

This chapter presents the **ACL** commands in alphabetical order.

Each entry includes the following information:

- Command name.
- Operative mode: DIRECT and/or EDIT.
- Command format.
- Complete description of the command.
- Examples of use.
- Additional notes, including references to related commands and subjects.

A / <Ctrl>+A DIRECT

Format:

`A [`*`prog`*`]`

`<Ctrl>+A`

Where: *prog* is a running program.

Description:

`A`
or `<Ctrl>+A` Immediately aborts all running programs and stops movement of axes.

<Ctrl>+A is the fastest software method for stopping program execution and halting the movement of all axes.

`A` *`prog`* Aborts the running of the specified program only.

Examples:

- `A` Aborts all programs.
- `A NEW` Aborts program NEW.

Notes:

The command <Ctrl>+A does not require <Enter> for execution.

The command A requires <Enter> for execution.

ANDIF

Format: `ANDIF` *`var1 oper var2`*

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

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

Example: ■

```
IF A=B                    If the values of A and B are equal,
   ANDIF C>2              and if the value of C is greater than 2,
   CLOSE                  close the gripper;
ELSE                      If any of the conditions is not true,
   OPEN                   open the gripper.
ENDIF                     End of conditional routine.
```

Note: Refer to the IF command.

APPEND DIRECT

Format: `APPEND`

Description: APPEND loads the contents of a backup file in the host computer to the controller's user RAM via the RS232 channel.

APPEND is similar to the RECEIVE command, but does not erase or modify existing programs.

The file must be in the format generated by a SEND command.

When the APPEND command is executed, the following occurs:

- New programs are accepted.
- New variables are accepted.
- New positions are accepted.
- Coordinate values will be assigned to defined positions whose coordinate values have not yet been set.

Note: The **ATS** Backup Manager performs the SEND, RECEIVE and APPEND procedures. Use that menu to backup and restore user RAM.

Refer to the chapter on the Backup Manager in the ***ATS** Reference Guide*.

Refer also to the SEND and RECEIVE commands.

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].

-
  ```
  ATTACH OFFB
  ```

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

CLOSE DIRECT/EDIT

Format: `CLOSE [var]`

Where: *var* is a variable or constant 0 ≤ *var* ≤ 5000

Description:

`CLOSE`	Closes electric 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.

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

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:

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

Notes: Refer to the OPEN and JAW commands.

Refer also the gripper parameters in Chapter 7.

CLR

Format: `CLR` *n*

Where: *n* is an encoder number, 1 ≤ *n* ≤ 11, or *.

Description:

`CLR` *n*	Clears (sets to zero) the values of a specific encoder.
`CLR *`	Clears the values of all encoders.

Warning! CLR spoils the robot arm's home reference, and alters all other positions as a result. Use with caution.

Example:

■ `CLR 3`	Clears encoder 3.

CLRBUF DIRECT/EDIT

Format:

`CLRBUF[A/B]`

`CLRBUF` *`axis`*

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

Description:

`CLRBUF` Empties the movement buffer of all axes, thereby aborting current and remaining movement commands. Can be used to stop the robot or axes upon event, and to continue the program with other commands.

`CLRBUFA` Empties the movement buffer of group A.
`CLRBUFB` Empties the movement buffer of group B.

`CLRBUF` *`axis`* Empties the movement buffer of a specific axis in group C.

Examples:

■
```
CLRBUF                      Empties the movement buffer of all axes.
```

■
```
IF IN[3]=1                  If input 3 is on;
   STOP MAIN                stop program MAIN;
   CLRBUFA                  clear all remaining MOVE commands
                               from the buffer of group A;
   MOVE HOME                move to position HOME.
ENDIF
```

CLRCOM

Format: `CLRCOM` *n*

`CLRCOM 0`

Where: *n* is the RS232 communication port, 1 ≤ *n* ≤ 8;
0 = all RS232 communication ports.

Description: `CLRCOM` *n* Clears the buffers of the specified RS232 port.

`CLRCOM 0` Clears the buffers of all RS232 communication ports.

This command can be used to reset the communication ports when an error, such as XOFF without a subsequent XON, interrupts or halts RS232 communication.

Example: ■ `CLRCOM 2` Clears the buffers of RS232 port COM2.

Note: Refer to the SENCOM command.

COFF DIRECT

Format: `COFF[A/B]`

`COFF` *n*

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

Description: Disables servo control for all axes or for a specified group or axis.

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

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.
- The controller's Emergency switch is pressed.
- The controller detects an impact or thermic error condition (as determined by system parameters).

COFF must be activated before you change parameters values.

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

When COFF 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.
■ `COFFB`	Control OFF for group B axes.
■ `COFF 10`	Control OFF for axis 10 in group C.

Note: Refer to the CON command.

DIRECT CON

Format:

```
CON[A/B]

CON n
```

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

Description:

Enables servo control for all axes or for a specified group or axis.

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.

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.

CONFIG DIRECT

Format: `CONFIG [?]`

Description: The CONFIG command allows you to perform a complete configuration of the controller. During the configuration the system displays the existing values [in brackets] and allows you to change them, as shown in the example below. If you do not want to change a setting, accept it by pressing <Enter>.

Warning! This command erases all programs, variables and positions in user RAM !

`CONFIG`	Activates the file used for configuring the controller. Allows you to define: number of inputs and outputs; number of servo axes; type of robot; size of memory reserved for user defined programs and program lines, and user defined variables, positions and comments.
`CONFIG ?`	Displays the current configuration

Examples: ■

```
>config
!!!WARNING ALL USER PROGRAMS WILL BE ERASED.
ARE YOU SURE ??? [YES/NO] > yes
JOB KILLING PHASE .........>
ENTER NUMBER OF INPUTS [16] (0-16) >
ENTER NUMBER OF OUTPUTS [16] (0-16) >
ENTER NUMBER OF ENCODERS [8] (0-11) >
ENTER NUMBER OF DACS [8] (0-11) >
ENTER NUMBER OF AUXILIARY RS232 PORTS[0] (0-8) >
WHICH TYPE OF ROBOT (0-5-7) [5] (0-7) >
ENTER NUMBER OF SERVO LOOPS, GROUP A [5] (5-8) >
SERVO GRIPPER INSTALLED AT AXIS [6] (0-8) >
ENTER NUMBER OF SERVO LOOPS, GROUP B [2] (0-2) >
ENTER TOTAL NUMBER OF SERVO LOOPS [8] (8-8) >
ENTER AMOUNT OF USER RAM IN KBYTES [128] (16-128) >
IS VISION BOARD PRESENT? [NO] >
ENTER NUMBER OF USER PROGRAMS [150] >
ENTER NUMBER OF USER PROGRAM LINES [3000] >
ENTER NUMBER OF USER VARIABLES [600] >
ENTER NUMBER OF USER POINTS , GROUP A [2380] >
ENTER NUMBER OF USER POINTS , GROUP B [2380] >
ENTER NUMBER OF USER POINTS , GROUP C [0] >
ENTER NUMBER OF USER COMMENTS [550] >
Performing configuration, please wait 10 seconds
```

```
>
>
O.K.
>
```

The system displays the existing (default) values, which you may change in accordance with the following:

- The maximum number of inputs and outputs is 16.
- The maximum number of encoders and DACS is 11, which is the maximum number of axes.
- When prompted for the type of robot, your options are as follows:
  - 0: Separate axes, kinematics of the arm unknown, no XYZ calculations, no HOME routine.
  - 5 : Compatible with **SCORBOT-ER V** and **SCORBOT-ER Vplus** kinematics.
  - Type 7 : Compatible with **SCORBOT-ER VII** kinematics.
- The number of axes for groups A and B are user definable. When the robot is defined as Type 5, group A must be the robot and include a minimum of 5 axes.
- If a servo gripper is being used, it must be installed as the next available axis following group A; for example, if group A includes 5 axes, the gripper must be installed as axis 6. If no servo gripper is installed, enter 0 as the gripper axis. You can then use axis 6 for driving other servo devices.
- The total number of servo loops cannot be less than the number of axes defined for groups A and B and gripper, and cannot exceed the number of encoders and DACs. Any remaining axes are assigned to group C, which always contains independent axes.
In this example the default settings are: 5 axes in group A, 2 axes in group B, and gripper, totaling 8 axes.
- The standard size of user RAM is 128Kb.
Controllers manufactured in 1990 and earlier may have less than 128Kb. If you are not sure about the quantity of RAM in your controller, contact your product representative.
- Vision board is for future use only. Do not try to answer positively.
- The number of user defined programs, program lines, variables, positions, and comments depends upon the memory size and the distribution of all these items.

  Refer to Chapter 6 for details of the memory required for each item, and calculate according to your needs.

INIT EDITOR is automatically executed during configuration.

- The following is an example of a current configuration report. The values in this example result from the configuration in the example given above.

```
>config ?
******* CURRENT CONFIGURATION IS :
INPUTS - 16
OUTPUTS - 16
ENCODERS - 8
ANALOG OUTPUTS - 8
AUXILIARY PORTS - 0
ROBOT TYPE - 5
SERVO AXIS GROUP A - 5
SERVO GRIPPER - 6
SERVO AXIS GROUP B - 2
TOTAL NUMBER OF SERVO AXIS 8
_128 KBYTE OF USER BATTERY BACKED UP MEMORY INSTALLED.
USER PROGRAMS - 150
USER PROGRAM LINES - 3000
USER VARIABLES - 600
USER POINTS , GROUP A - 2380
USER POINTS , GROUP B - 2380
USER POINTS , GROUP C - 0
USER COMMENTS - 550
>
```

CONTINUE

Format: `CONTINUE` *`prog`*

Where: *prog* is a suspended program

Description: Resumes execution of program *prog* from the point where it was previously suspended by the SUSPEND command.

Example:

- `CONTINUE ALPHA` Resumes execution of program ALPHA.

Note: Refer to the SUSPEND command.

COPY DIRECT

Format: `COPY prog1 prog2`

Where: *prog1* is an existing user program.

Description: Copies *prog1* to a new program named *prog2*.
Two copies of the same program now exist under different names.

If the name *prog2* is already in use, a warning message will appear.

Example:

- `COPY ALPHA BETA` Copies user program ALPHA to program BETA.

DEFINE

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.

DEFP DIRECT/EDIT

Format: `DEFP[A/B] pos`

`DEFPC pos n`

Where: *pos* is a user defined name;
n is an axis in group C.

Description: Defines a position for a specific axis control group. When a position is defined, controller memory is reserved for the position's coordinate values which will subsequently be recorded or set.

`DEFP pos`
`DEFPA pos`
	Defines a position for axis control group A.

`DEFPB pos`	Defines a position for axis control group B.
`DEFPC pos n`	Defines a position for an axis in group C.

If a group is not specified for the position, group A is assumed. Once a position 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.

Examples:

■ `DEFP S`	Defines a position named S for group A.
■ `DEFPA BF3`	Defines a position named BF3 for group A
■ `DEFPB DD`	Defines a position named DD for group B.
■ `DEFPC P85 9`	Defines a position named P85 for group C axis 9.

Note: This command does not create a program line.

Format: `DEL`

Description: Erases the last displayed line in a program which is being edited.

Example: ■

```
190: LABEL 1
191: MOVE 10
192:?DEL
```

Erases the command in line 191.

DELAY **EDIT**

Format: `DELAY var`

Where: *var* is a variable or constant.

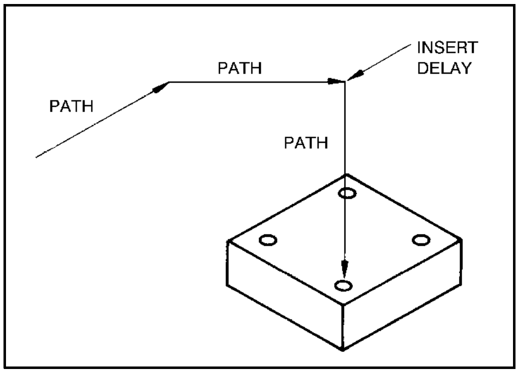
Description: Delays the execution of a program

Var is defined in hundredths 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 taught, 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.

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 erase a position which is in use.

DELP cannot delete individual positions within vectors.

UNDEF deletes 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.

Notes: This command does not create a program line.

Refer to the UNDEF command.

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 erase 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.

DIM

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 LOCV[20]` Creates a variable array named LOCV containing 20 private variables, LOCV[1] . . . LOCV[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.

DIRECT/EDIT

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.

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.

Examples:

■ `DIMP PICK[30]`	Creates a vector for group A containing 30 positions named PICK[1] . . . PICK[30].
■ `DIMPB BB[10]`	Creates a vector for group B containing 10 positions named BB[1] . . . BB[10].
■ `DIMPC CNV[25] 11`	Creates a vector for axis 11 containing 25 positions named CNV[1] . . . CNV[25].

Note: 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 numbers 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
DEMO        :           : 1        : 5
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.

DISABLE

Format:

```
DISABLE {IN/OUT} n
```

```
DISABLE ?
```

Where: IN is an input;
OUT is an output;
n is the I/O index, $1 \le n \le 16$

Description:

`DISABLE IN` *n* `DISABLE OUT` *n*	Disconnects the physical input or output from normal system control.
`DISABLE ?`	Displays all disabled inputs and outputs.

When an input or output is disabled, its last state remains unchanged. However, the FORCE command can be used to alter its state.

To restore normal system control of a disabled input or output, use the ENABLE command.

Examples:

■ `DISABLE IN 8`	Disconnects input 8 from normal system control.
■ `DISABLE OUT 12`	Disconnects output 12 from normal system control.

Note: Refer to the ENABLE and FORCE commands.

DO DIRECT

Format: `DO` *`editcom`*

Where: *editcom* is an EDIT mode command.

Description: Performs any of the following EDIT mode commands in DIRECT mode.

```
POST        CLRCOM
QPEND       PRCOM
PRINTLN     PRLNCOM
READ        READCOM
STOP        SENCOM
```

Examples:

- `DO CLRCOM 2` Immediately resets communication port 2.
- `DO PRINTLN` Immediately inserts a carriage return and line feed.

ECHO

Format: `ECHO`

Description: In ECHO mode, all characters that are transmitted to the controller are displayed on the screen. This is the default mode.

In NOECHO mode the transmitted characters are not displayed.

Note: Refer to the NOECHO command.

EDIT DIRECT

Format: `EDIT` *`prog`*

Where: *prog* is any user program.

Description: Activates the EDIT mode and calls up a user program named *prog* . If *prog* is not found, the system automatically creates a new program of that name, and waits for the user's confirmation.

To quit the EDIT mode, and return to DIRECT mode, use the command:

```
EXIT
```

Example: ■

```
>edit ALPHA
Welcome to ACL editor, type HELP when in trouble

PROGRAM ALPHA
*****************
127:?                             The editor is now ready to receive program lines.
```

Notes: Refer to the EXIT command.

Refer to Chapters 1 and 2 for descriptions of editing commands.

Refer to Chapter 4 for information on reserved names for user programs.

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.

ENABLE DIRECT

Format: `ENABLE {IN/OUT} n`

Where: IN is an input;
OUT is an output;
n is the I/O index, $1 \le n \le 16$

Description:

`ENABLE IN n`
`ENABLE OUT n`
	Restores normal system control of the specified input or output.

By default, all the inputs and outputs are enabled.

Examples:

■ `ENABLE IN 8`	Reconnects input 8 to normal system control.
■ `ENABLE OUT 12`	Reconnects output 12 to normal system control.

Note: Refer to the DISABLE command.

END

Description: The system **automatically** writes END as the last line of a program.

You do not need to enter this command.

ENDFOR EDIT

Format: ENDFOR

Description: Required companion to the 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

Format: `ENDIF`

Description: Required companion to the IF command.

End the subroutine to be executed by the IF command.

Example: ■

```
IF XYZ=1                If the first condition and the second conditions 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

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`	Terminates SHOW ENCO, LIST, and SEND commands.

EXACT

Format: `EXACT [OFF]{A/B/C}`

Determines the accuracy of the commands which are used for sequential execution of operations in a program:

```
MOVED          MOVELD
MOVESD         MOVECD
```

The EXACT and EXACT OFF modes are applied separately to each axis control group.

`EXACT {A/B/C}`

Enables the EXACT mode for group A, group B or group C.

When a movement command (with D suffix) is executed in EXACT mode, the axes reach the target position accurately (within a given position error tolerance).
Movement *duration*, if specified in the movement command, is ignored when the command is executed in EXACT mode.

`EXACT OFF{A/B/C}`

Disables the EXACT mode for group A, group B or group C.

When a movement command (with D suffix) is executed in EXACT OFF mode, the axes reach the target position within a specified *duration*. Position accuracy is not guaranteed.

By default, all groups are in EXACT mode.

Examples:

- `EXACT A` EXACT mode ON for group A.
- `EXACT OFFA` EXACT mode OFF for group A.

Notes: Parameter 260+*axis* determines the position error tolerance.

Refer to the commands MOVED, MOVESD, MOVELD, and MOVECD.

EXIT EDIT

Format: `EXIT`

Description: Quits EDIT and checks the logic of the program. Searches for errors, such as FOR commands without ENDFOR, IF without ENDIF, and GOTO without a proper LABEL.

If an error is found, a message is displayed:

```
PROGRAM NOT VALID
```

And, when possible, the cause of the error is indicated.

If no errors are found, the following message is displayed:

```
PROGRAM IS VALID
```

EXIT returns the controller to the DIRECT mode.

FOR

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
  ```

FORCE DIRECT

Format:

```
FORCE {IN/OUT} n {0/1}
FORCE ?
```

Where:
IN is an input;
OUT is an output;
n is the I/O index;
0=off; 1=on

Description:

`FORCE` Forces the specified input or output to the specified state.

This command is operative only for I/Os which have been disabled by the DISABLE command.

`FORCE ?` Displays a list of all forced inputs and outputs, and their state.

Examples:

-
  ```
  DISABLE IN 5
  FORCE IN 5 1
  ```
  Activates input 5 to ON state.

-
  ```
  DISABLE OUT 11
  FORCE OUT 11 0
  ```
  Activates output 11 to OFF state.

-
  ```
  >force ?
  INput[5]=1
  OUTput[11]=0
  ```
  Input 5 is in forced ON state.
  Output 11 is in forced OFF state.

Note: Refer to the DISABLE command.

DIRECT FREE

Format: FREE

Description: Displays a list of the available memory in user RAM:

- Available program lines
- Available variables
- Available points of group A
- Available points of group B
- Available points of group C
- Available bytes for comments

Example: ■

```
>FREE
484 LINES ARE FREE
- - - - - - - - - - - - - - - -
63 VARIABLES ARE FREE
- - - - - - - - - - - - - - - -
310 POINTS OF GROUP A ARE FREE
- - - - - - - - - - - - - - - -
308 POINTS OF GROUP B ARE FREE
- - - - - - - - - - - - - - - -
11 POINTS OF GROUP C ARE FREE
- - - - - - - - - - - - - - - -
900 BYTES ARE FREE for comments.
>
```

GET EDIT

Format: GET *var*

Where: *var* is a user defined 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.

GETCOM

Format: `GETCOM` *`n var`*

Where: *n* is an RS232 communication port, 1 ≤ *n* ≤ 8;
var is a variable.

Description: Companion to the SENCOM command.

Receives one byte from the specified RS232 port.

The value of the byte is stored in the specified variable.

Example: ■

```
    PROGRAM WAIT1
    *****************
LABEL    1
GETCOM 1 RCV
PRINTLN "RECEIVING ASCII CODE: "RCV
GOTO 1
END
```

This program waits for a character to be received on RS232 port COM1, and then displays its value on the screen.
If the character A (ASCII 65) is pressed, the following is displayed on the screen:

```
RECEIVING ASCII CODE : 65
```

Note: Refer to the SENCOM 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.

GOSUB

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
MOVED 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 ≤ *n* ≤ 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:

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

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

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

  This program is executed 500 times and then stops.

Note: Refer to the LABEL command.

HELP

Format: `[DO] HELP`

Description: When in DIRECT mode:

HELP provides an on-line help screen for DIRECT commands, and DO HELP provides on-line help screen for EDIT commands.

When in EDIT mode:

HELP provides a list and brief explanations of all EDIT commands. This command does not create a program line.

HERE DIRECT/EDIT

Format: `HERE pos`

Where: *pos* is a defined position.

Description: Records an absolute position, in **joint** (encoder) values, according to the current location of the axes.

Pos must first be defined using the DEFP or DIMP command.

Examples:

- `HERE POINT` Records the coordinates of position POINT.

- `DIMP P[20]`
  `HERE P[5]` Defines a vector named P containing 20 position; records vector position 5.

DIRECT/EDIT

HERER

Format:

`HERER` *`pos2`* — DIRECT mode only.

`HERER` *`pos2 pos1`*

Where: *pos1* is a recorded position for any group;
pos1 and *pos2* are defined for the same group.

Description:

HERER allows you to record a position relative to another position, or relative to the current position of the robot.

`HERER` *`pos2`*

Records the offset coordinates of *pos2* relative to the current position in **joint** (encoder) values.

You are prompted to enter the offset values as shown in the example below.

Pos2 will always be relative to the current position.

`HERER` *`pos2 pos1`*

Records the offset coordinates of *pos2* relative to *pos1* in joint values. *Pos1* must be recorded before *pos2*.

Pos2 is always relative to *pos1*, moving along with and maintaining its offset whenever *pos1* is moved.

Examples:

■

```
>DEFP AA
>HERER AA
   1 -- [.] > 0
   2 -- [.] > 500
   3 -- [.] > 250
   4 -- [.] > 0
   5 -- [.] > 0
```

Defines and records relative position AA.
AA will always be relative to the robot’s current position by user defined values:
0 encoder counts in base
500 encoder counts in shoulder
250 encoder counts in elbow
0 encoder counts in pitch
0 encoder counts in roll

The value in brackets [ ] indicates the offset values last entered for this position. If the bracket is empty, no value is in memory.

■

```
>DEFPB PST
>HERER PST
   7 -- [.] > 100
   8 -- [.] > 0
```

Defines and records relative position PST for group B. PST will always be relative (by 100 encoder counts on axis 7) to the current position of the device connected to axes 7 and 8.

■

```
HERE BB
(move robot)
HERER AA BB
```

Records position BB, then records position AA as relative to position BB by the offset values which are automatically entered by this command.

■

```
DIMPB PLT[5]
HERE PLT[1]
(move device)
HERER PLT[2] PLT[1]
```

Records vector PLT for group B. Records position PLT[1], then records PLT[2] as relative to position PLT[1] by the offset values which are automatically entered by this command.

HOME / HHOME

Format: `HOME [`*n*`]`

`HHOME` *n*

Where: *n* is an axis, 1 ≤ *n* ≤ 11.

Description: The HOME command activates the internal system procedure HOME.

`HOME`	Drives all robot axes to their home position by searching for a microswitch on each axis. The home search is performed only if Robot Type 5 or 7 was entered during configuration.
`HOME` *n*	Drives the specified axis to its home position, by searching for a microswitch. The HOME *n* command allows you to create a homing program suitable for a particular configuration.
`HHOME` *n*	Drives the specified axis to its home position by searching for a hard stop. The HHOME command is used for a device, such as a slidebase, which does not have a microswitch.

The robot and peripheral axes should be homed at the beginning of each working session.

Activating HOME aborts all running user programs and activates servo control (CON).

During the robot homing, the robot joints move and search for their home positions, one at a time, in the following sequence: shoulder, elbow, pitch, roll, base, gripper. The following message is displayed:

```
WAIT!! HOMING...
```

If all axes reach their home position, a message is displayed:

```
HOMING COMPLETE (ROBOT)
```

The system records **Position 0** at the end of homing. This position contains the coordinates of the robot after it has been homed; the coordinate values are not necessarily 0.

If the homing process is not completed, an error message identifying the failure is displayed:

```
*** HOME FAILURE AXIS 6
```

Examples:

■ `HOME 7`
`HOME 8`
`HOME 9`
	Searches for a microswitch home on axes 7, 8, and 9.

■ `HHOME 7`	Searches for hard stop home on linear slidebase connected to axis 7. **Note:** The moving base must be near enough to the mechanical end stop of the LSB for the homing to succeed.

Notes:

To run the robot HOME program from the teach pendant, key in:

[Run] 0 [Enter]

The power used for driving the motor in an HHOME command is determined by the system parameters 200+*axis*. Make sure this DAC value will not damage the connected axes!

Refer to the homing parameters in Chapter 7.

IF

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

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

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

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

INIT DIRECT

Format:

```
INIT CONTROL
INIT EDITOR
INIT PROFILE
```

INIT CONTROL

Description: Resets all system control parameters.

INIT CONTROL must be executed after any change in system control parameters.

Note: Refer to the LET PAR command.

INIT EDITOR

Description: Initializes controller's user RAM configuration—programs, positions and variables. It does not affect parameters.

Warning! This command erases the contents of user RAM.

Note: The CONFIG command automatically performs this operation.

INIT PROFILE

Description: Initializes the velocity profiles (both paraboloid and trapezoid) according to the value of parameter 76.

INIT PROFILE must be executed after any change in parameter 76.

Note: Refer to Chapter 7 for an explanation of parameter 76.

INT_ON / INT_OFF

Format:

```
INT_ON axis1 [axis2 ... axis4]
INT_OFF axis1 [axis2 ... axis4]
```

Where: *axis1* . . . *axis4* are servo axes.

Description:

INT_ON enables integral feedback control for the servo axis or axes specified.

INT_OFF disables integral feedback control for the servo axis or axes specified, leaving only proportional and differential feedback control.

Up to four axes can be specified in one command. The switching occurs at run time.

Disabling integral feedback control during movement can be useful for canceling overshoot. Reestablishing integral feedback at the end of movement will bring the robot arm to its exact target position.

Disabling integral feedback control is also useful for limiting the power applied to motors at the end of a movement, for instances in which an obstacle near the target position does not allow the arm to reach its target. For example, if the robot’s task is to place a cube on the table, and the cube’s dimensions are not precise (too big): with INT_ON, the DAC value will increase until it reaches the maximum, resulting in a Thermic Protection error; with INT_OFF, the DAC value remains constant, even if the robot is not precisely at the target position.

By default, all axes are in INT_ON mode.

Example: ■

```
MOVE A 300
INT_OFF 1 2 3 4
DELAY 300
INT_ON 1 2 3 4
```

Note: Make sure to include an underscored space between INT and ON/OFF.

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 hundredths 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`	Opens the gripper to 40 percent of its opening.
■ `JAW 0`	Closes the gripper.

Note: Refer to the CLOSE and OPEN commands.

EDIT L

Format: L *line1 line2*

Description: Displays a list of program lines, from the first line specified to the second line specified.

Example: ■

```
16:?L 3 13
 ****** listing 3 to 13*****
 3: GOSUB MVMAX
 4: IF MVMAX
 5: IF VA >= VB
 6: MOVE 0
 7: DELAY 1
 8: SET TI=LTA - LTB
 9: IF TI > 100
10: MOVE 00 TI
11: ELSE
12: MOVE 00
13: ENDIF
 **** End of listing ****
```

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.

LET PAR

Format:

```
LET PAR n var
LET PAR n=var
```

Where: *n* is a parameter number, and
var is a variable or constant.

Description: Sets the value of system parameter *n* to *var* .

After you have set new system parameters, you must put them into effect by issuing the command:

```
INIT CONTROL
```

If you have changed parameter 76, you must issue the command:

```
INIT PROFILE
```

Examples:

-
  ```
  LET PAR 21=100
  INIT CONTROL
  ```
  Sets parameter 21 to 100.

-
  ```
  LET PAR 295 50
  INIT CONTROL
  ```
  Sets parameter 295 to 50.

-
  ```
  LET PAR 76 5
  INIT PROFILE
  ```
  Sets parameter 76 to 5.

Notes: *Warning! Only experienced users should attempt parameter manipulation.*

Refer to Chapter 7 for information on system parameters, and heed all warnings given there.

LIST DIRECT

Format: `LIST [`*`prog`*`]`

Where: *prog* is a user defined program.

Description:

`LIST` *`prog`*	Displays all lines of user program *prog* .
`LIST`	Displays all lines of all user programs.

To stop the display, use <Ctrl>+C.

Example: ■ **>LIST AAA** 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: 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.

DIRECT LISTP

Format: LISTP

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

Example: ■

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

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

LISTPV DIRECT

Format:

`LISTPV` *`pos`*

`LISTPV POSITION`

Description:

`LISTPV` *`pos`*

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

X,Y and Z coordinates are expressed in tenths of millimeters, and indicate the distance from the robot’s point of origin— the center and bottom of the robot’s base—to the TCP (tool center point = gripper tip). P (pitch) and R (roll) values are in tenths of degrees.

`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.

Example:

■

```
>LISTPV P1
Position P1

1:0 2: 5     3: 13926     4:0          5:0
X: 5197      Y:0          Z: 9963      P: 623       R:-3
```

P1 is a robot position; both joint and Cartesian values are also displayed. For example: Z: 9963 means Z = 996.3 mm;
P: -623 means P = -62.3°.

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[11]
TIME
LTA
LTB
MFLAG
ERROR
OUT[16]
ANOUT[11]

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

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

LSON / LSOFF DIRECT/EDIT

Format:

```
LSON
LSOFF
```

Description:

`LSON`	Connects the **homing microswitches** on the axes to the controller's input sensors. When LSON mode is active, the inputs are disabled. All references to inputs (including system variable IN[*n*]) actually refer to the home switches.
`LSOFF`	Disconnects the homing microswitches from the controller's input sensors. When LSOFF mode is active, the **inputs** are enabled.

HOME automatically switches to LSON and back to LSOFF.

LSOFF is the default mode; inputs are enabled.

Use the command SHOW DIN to display the state of either the inputs or the microswitches.

Examples:

■ `LSON` `SHOW DIN`	Disables the inputs, and displays the status of all home switches.
■ `LSOFF` `SHOW DIN`	Enables the inputs, and displays the status of all inputs.

Note: Refer to the command SHOW DIN.

MOVE / MOVED

Note that execution of MOVE is not synchronized with program flow! The MOVED command 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 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. *Duratio*n 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 axis reaches 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 within the required accuracy, regardless of duration.

MOVE, MOVED Summary

`MOVE`	Easy to program, but cannot guarantee sequentiality and 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 reached within the tolerated accuracy. You will notice the axes pausing 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 position 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 a 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 = gripper tip) 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 SPEED command defines the speed of the TCP. The duration of movement is thus determined by the path length and the SPEED definition.

The starting position, *pos1*, and *pos2* should define a circle. These 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.

-
  ```
  SPEED 20
  MOVEC 2 1
  ```
  Moves along a circular path from current position to position 2 via position 1, at speed rate 20.

Note: Refer to the SPEED command.

DIRECT/EDIT MOVEL / MOVELD

Format: `MOVEL` *`pos1`* `[`*`duration`*`]`

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

Description: Moves the robot’s TCP (tool center point = gripper tip) 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 SPEED command.

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

All other aspects of this command 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 SPEED 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 a position vector;
n1 is the index of the first position;
lastpos 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.

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.

One movement profile is applied to the entire movement. Trajectory acceleration and deceleration occur only at the beginning and end of the full movement.

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

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.

MPROFILE

Format:

```
MPROFILE PARABOLE {A/B/C}
MPROFILE TRAPEZE {A/B/C}
```

Description: Assigns a movement profile to a specific axis control group.

For better path performance, two path control profiles are available: **paraboloid** and **trapezoid**. The PARABOLE command causes the motors to accelerate slowly until maximum speed is reached, then decelerate at the same rate. The TRAPEZE command causes the motors to accelerate and decelerate quickly at the start and end of movement, with a constant speed along the path. See the diagram below.

You can assign different control profiles to different control groups. For example: paraboloid profile for group A, trapezoid profile for group B.

Paraboloid profile is most suitable for applications which do not require constant speed, since it does not overstress the motors.

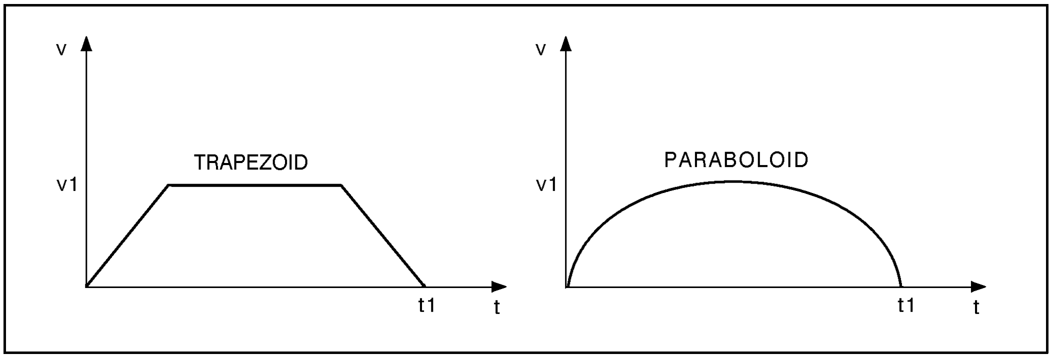
Trapezoid profile is most suitable for applications such as welding. spay painting, or gluing, which require a constant speed during part of the path.

By default, the paraboloid profile is active for all groups.

Example:

- `MPROFILE TRAPEZE A` Changes robot movement profile to TRAPEZE.

Note: Minimum velocity for accelerating and decelerating is determined by parameter 76.



NOECHO DIRECT

Format: `NOECHO`

Description: When in NOECHO mode, characters transmitted to the controller are not displayed on the screen.

The ECHO command cancels the NOECHO mode.

By default, the controller is in ECHO mode.

Note: Refer to the ECHO command.

NOQUIET

Format: NOQUIET

Description: During program execution, all DIRECT commands within the program (that is, DIRECT commands preceded by @) are displayed as they are executed.

This is the default mode.

Note: Refer to the QUIET command.

OPEN DIRECT/EDIT

Format: `OPEN [`*`var`*`]`

Where: *var* is a user defined variable or constant 0 *var* <5000.

Description:

`OPEN`	Opens electric gripper until end of gripper motion.
`OPEN` *`var`*	A variable or a constant is set to the gripper DAC to maintain drive to the gripper motor for additional grasping force. The greater the value of *var*, the stronger the drive force.

The OPEN command disconnects the gripper from the servo control loop.

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 to the value of PRESS

Notes: Refer to the CLOSE and JAW commands.

ORIF

Format: `ORIF` *`var1 oper var2`*

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

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

P EDIT

Format: P

Description: Takes the editor to the preceding line in the program currently being edited.

PEND / POST

Format:

`PEND` *`var1`* `FROM` *`var2`*

`POST` *`var3`* `TO` *`var2`*

Where: *var1* and *var2* are user defined variables (*var2* must be global); *var3* is a variable or a constant.

Description:

The PEND and POST commands are used for synchronizing the simultaneous execution of programs.

When a program encounters a PEND *var1* FROM *var2* command, one of the following occurs:

- If *var2* has a value of zero, program execution is suspended until another running program “sends” a non-zero value by means of the POST *var3* TO *var2* command.
- If *var2* has a non-zero value, that value is assigned to *var1* and the value of *var2* is set to zero.

Example: ■

```
   PROGRAM DOACT
   ****************
GLOBAL SIGN
DEFINE VALUE
SET SIGN=0
PEND VALUE FROM SIGN
RUN ACT
END
```

The execution of program DOACT will be suspended until program SEND is activated and sets the value of SIGN to 1.

```
   PROGRAM SEND
   ****************
POST 1 TO SIGN
END
```

PRCOM EDIT

Format: `PRCOM n arg1 [arg2 arg3]`

Where: *n* is the RS232 communication port, 1≤*n*≤8, and
arg is a variable or a string within quotation marks (“ ”).

Description: Sends strings and variable values to the specified RS232 port.

The text following PRCOM *n* may contain up to 30 characters and spaces, not including the quotation marks. The text may contain a total of 3 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 and 20 characters are treated, respectively, as two and three arguments.

Examples:

```
PRCOM 6 "TESTING"
```

The text “TESTING” will be transmitted to RS232 port COM6.

■
```
SET X=7
PRCOM 5 "PRICE IS " X " DOLLARS"
```

The text “PRICE IS 7 DOLLARS” will be transmitted to RS232 port COM5.

Note: Refer to the PRLNCOM 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: 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 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.

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.

PRIORITY

Format: `PRIORITY` *`prog var`*

Where: *prog* is a user program
var is a variable or a constant

Description: The PRIORITY command sets the priority of *prog* to the value of *var* .

Priorities range from 1 to 10, with 10 as 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 power on), all programs are assigned a priority of 5.

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.

Example: ■ `PRIORITY PALET 7` Assigns program PALET a priority of 7.

Note: Refer to the RUN and DIR commands.

PRLNCOM EDIT

Format: `PRLNCOM` *`n arg1`* `[`*`arg2 arg3`*`]`

Where: *n* is the RS232 communication port, 1≤*n*≤8, and
arg is a variable or a string within quotation marks (“ ”).

Description: Companion to READCOM command.

Sends strings and variable values to the specified RS232 port.

Same as the PRCOM command, but adds a carriage return *after* sending the text to the RS232 port.

The text following PRLNCOM *n* may contain up to 30 characters and spaces, not including the quotation marks. The text may contain a total of 3 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 and 20 characters are treated, respectively, as two and three arguments.

Example: ■

```
PRLNCOM 7 "THE VALUE IS " VAL[I]
```

The text “THE VALUE IS” and the value of variable VAL[I] will be transmitted to RS232 port COM 7, followed by a carriage return .

If, for example, the value of VAL[I] is 26, the string “26” (not ASCII character 26) will be sent.

Note: Refer to the PRCOM and READCOM commands.

QPEND / QPOST

Format:

`QPEND` *`var1`* `from` *`var2`*

`QPOST` *`var3`* `to` *`var2`*

Where: *var1* is a variable
var2 is a global variable array
var3 is a variable or constant

Description:

`QPEND` Takes values from a queue in the same order they were entered by the QPOST command.

`QPOST` Queues the values to be processed.

If the queue is exhausted, QPEND suspends program execution until a QPOST command enters a value.

The maximum size of the queue is equal to the dimension of the *var2* array minus 1. If the queue is full, QPOST suspends program execution until a QPEND command takes a value from the queue.

A queue must be initialized before use by setting all its elements to zero.

Example: ■

```
   PROGRAM INITQ
DIMG QUEUE[10]
DEFINE I
FOR I=1 TO 10
 SET QUEUE[I]=0
ENDFOR
END
```

Defines and initializes the queue.

```
   PROGRAM DOACT
DEFINE VALUE
LABEL 1
QPEND VALUE FROM QUEUE
RUN ACT
GOTO 1
END
```

Takes a value from a queue.
Program ACT will run when values are deposited in QUEUE by the program SEND. If no value has been sent, DOACT will be suspended until the arrival of a value.

```
   PROGRAM SEND
QPOST 1 TO QUEUE
END
```

Puts a value in a queue.

QUIET DIRECT

Format: `QUIET`

Description: Cancels the NOQUIET mode.

When in QUIET mode, DIRECT commands within the program (those preceded by @) will **not** be displayed during the program's execution.

By default, the controller is in NOQUIET mode.

Note: Refer to the NOQUIET command.

READ

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.

READCOM EDIT

Format: `READCOM n, var`

Where: *n* is the RS232 communication port, 1 ≤ *n* ≤ 8, and
var is a variable.

Description: Companion to PRLNCOM command.

When a READCOM command from the specified port is encountered, it waits on line for a string which contains ASCII numbers followed by a carriage return. That numeric value is then assigned to the specified variable.

Example: ■

```
READCOM 1, RPART
IF RPART > 9999
   PRINTLN "CAN’T MANUFACTURE MORE THAN 9999 PIECES"
```

Note: Refer to the PRLNCOM command.

RECEIVE

Format: `RECEIVE [`*`prog`*`]`

Description: Loads data from a user backup file in the host computer to the controller's user RAM via the RS232 channel.

The file to be received must be in the format created by the SEND command.

`RECEIVE`

Warning! This command erases the contents of the controller's user RAM.

Accepts the contents of a backup file generated by the SEND commands.

After you enter the RECEIVE command, the controller responds with a warning that all user programs, positions and variables will be erased, and prompting you to confirm.

If your response is YES, the controller replies with the following message:

```
PLEASE SEND FILES
```

Refer to your terminal documentation for exact instructions on sending and receiving files.

`RECEIVE` *`prog`*

Accepts the contents of a backup file generated by the SEND commands.

Accepts only one program and inserts its contents into the *prog* specified. It does not affect the other programs and positions stored in the user RAM.

The host computer sends the file line by line to the controller. After each line the host computer waits for a colon ":" to be transmitted by the controller. This indicates that the next line can be sent.

The last line of the file to be transmitted must be the message:

```
(END)
```

To which the controller responds:

```
END OF LOADING
```

Notes: The **ATS** Backup Manager performs the SEND, RECEIVE and APPEND procedures. Use that menu to backup and restore user RAM.

Refer to the chapter on the Backup Manager in the ***ATS*** *Reference Guide*.

Refer also to the SEND command.

DIRECT

REMOVE

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.

RENAME DIRECT

Format: `RENAME  prog1 prog2`

Description: Changes the name of user program from *prog1* to *prog2*

If the name *prog2* already exists, the command is not executed, and an error message is displayed.

Once a program name has been changed, the original *prog1* no longer exists.

Example:

- `RENAME PAL NEW` Program PAL is now called NEW. Program PAL is no longer listed in the directory.

RUN

Format: `RUN` *`prog`* `[`*`var`*`]`

Where: *prog* is a user defined program name, and
var is a user defined 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.

S EDIT

Format: `S [`*`line_n`*`]`

Where: *line_n* is a program line number

Description:

`S`	Moves editor to the first line of the program currently being edited.
`S` *`line_n`*	Moves editor to the specified line of the program currently being edited.

SENCOM

Format: `SENCOM` *`n var`*

Where: *n* is the RS232 communication port, 1 ≤ *n* ≤ 8;
var is a variable or constant.

Description: Companion to the GETCOM command.

Sends one byte through the specified RS232 port.

The value of the byte is specified by a variable or a constant.

Example: ■

```
   PROGRAM ESC
   *********
DEFINE I
CLRCOM 2
FOR I=1 TO 5
   SENCOM 2, 27
   DELAY 20
ENDFOR
END
```

This program clears the buffers of RS232 port 2. It then sends 27, the ASCII code for <Esc>, five times to port 2.

Note: Refer to the GETCOM command.

SEND DIRECT

Format:

`SEND`	`SEND` *`prog`*
`SENDPROG`	`SENDVAR`
`SENDPOINT`	`SENDPAR`

Where: *prog* is a user defined program

Description:

SEND commands produce listings in a format compatible with the RECEIVE and APPEND commands. The listings produced by the SEND commands are displayed on the computer screen.

`SEND` Generates a listing of all user programs, variables and positions, and parameters. SEND serves to create a complete backup of user RAM.

`SEND` *`prog`* Generates a listing of the specified user program in a format compatible with the RECEIVE *prog* command.

`SENDPROG` Generates a listing of all user programs, variables, and positions.
SENDPROG serves to create a backup of user RAM, except for parameters.

`SENDVAR` Generates a listing of all user defined variables.

`SENDPOINT` Generates a listing of all user defined positions.

`SENDPAR` Generates a listing of all system parameters.

Notes:

The **ATS** Backup Manager performs the SEND, RECEIVE and APPEND procedures. Use that menu to backup and restore user RAM.

Refer to the chapter on the Backup Manager in the ***ATS** Reference Guide*.

Refer also to the SEND command.

Format:

`SET` *`var1`*`=`*`var2`*

`SET` *`var1`*`=NOT` *`var2`*

`SET` *`var1`*`=COMPLEMENT` *`var2`*

`SET` *`var1`*`=ABS` *`var2`*

`SET` *`var1`*`=`*`var2 oper var3`*

`SET` *`var`*`=PVAL` *`pos axis`*

`SET` *`var`*`=PVALC` *`pos coord`*

`SET` *`var`*`=PSTATUS` *`pos`*

Where: *var* is a variable;
var1 var2 and *var3* are variables or constants;
oper can be: + – * / COS SIN TAN ATAN EXP LOG MOD AND OR;
pos is a position;
axis is an axis number;
coord is a Cartesian coordinate: X, Y, Z, or P or R.

Description:

1. `SET` *`var1`*`=`*`var2`* — Assigns the value of *var2* to *var1*

2. `SET` *`var1`*`=NOT` *`var2`* — Assigns the **logical negative** value of *var2* to *var1* .

   If *var2* ≤ 0, then *var1* = 1;
   If *var2* > 0, then *var1* = 0.

3. `SET` *`var1`*`=COMPLEMENT` *`var2`*

   Assigns the **complement** value of *var2* to *var1* .

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

4. `SET` *`var1`*`=ABS` *`var`*`2` — Sets the **absolute** value of *var1* to *var2* .

5. `SET var1=var2 oper var3`

Where *oper* is one of the following: +, -, *, /, MOD: The mathematical function is performed on *var2* and *var3* and the results are stored in *var1* .

Where *oper* is: AND, OR: The bit AND/OR operation is performed on *var2* and *var3* and the result is stored in *var1*.

Warning! Before performing any trigonometric or logarithmic function, carefully read the following description. The controller is an integer machine, and fractional values must be scaled to integer values.

Where *oper* is a trigonometric function: COS, SIN, TAN: *var2* acts as a multiplier, and *var3* is in degrees.

The trigonometric function of *var3* is computed and then multiplied by *var2* . Remember that the controller uses integral arithmetic, so the multiplier must be large enough to give the expected accuracy.

Where *oper* is: ATAN, EXP, LOG: The value of *var3* is divided by 10000 before computation is performed. Then the function is applied, and the result is multiplied by *var2* .

The result of the ATAN function is expressed in radians.

6. `SET var=PVAL pos axis` Assigns *var* the joint value of the specified axis in the specified position.

7. `SET var=PVALC pos coord`

Assigns *var* one of the Cartesian coordinates of the specified position.

Pos must be a robot (group A) position.

Coord can be one of the following:
X, Y, Z, P, R .
X, Y, Z are specified in tenths of millimeter; P (pitch), R (roll) are specified in tenths of a degree.

8. `SET` *`var`*`=PSTATUS` *`pos`* — Assigns *var* a value according to the type of the specified position.

Position Type	= Value
Position defined, but coordinates not recorded	0
Absolute joint position	1
Relative by joints to another position	2
Relative by XYZ (Cartesian) offset to another position	3
Relative by joints to current position	12
Relative by XYZ (Cartesian) offset to current position	13

Pos cannot be defined as POSITION, which is reserved for the current coordinates of the robot.

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 =0.5; Multiply by 1000; A is set to 500.
- `SET ST=PSTATUS P1` — If P1 is an absolute (fixed) position, then ST will be assigned a value of 1.
- `SET ANOUT[3]=2500` — Sets the analog output value for axis 3 to 2500. (ANOUT[*n*] is a system variable.)
- `SET OUT[1]=1` — Turns on output 5. (OUT[*n*] is a system variable.)
- `SET CLOCK=TIME` — Assigns value of TIME to variable CLOCK. (TIME is a system variable.)

SETP DIRECT/EDIT

Format: `SETP pos2=pos1`

Where: *pos1* is a recorded position;
pos2 and *pos1* are defined for the same group.

Description: Copies the coordinate values and position type of *pos1* to *pos2*.
Both positions are now identical.

This command is useful for preparing *pos2* so that the SETPV command can be used to change one value of that position.

Examples:

■ `SETP POINT=PLACE` Position POINT is assigned the coordinate values and type of position PLACE.

■ `SETP 100=POSITION` Position 100 is assigned the coordinate values of the current robot position.

Format:

`SETPV` *`pos`* DIRECT mode only.

`SETPV` *`pos axis var`*

Where: *pos* is a defined robot (group A) position;
axis is an axis number;
var is a variable or a constant.

SETPV

Description: Records an absolute joint position according to user defined values.

You are prompted to provide values for each of the joint coordinates of the specified position, in the following format:

```
SETPV P
   1 -- [100] >
   2 -- [130] >
   3 -- [250] >
   4 -- [120] >
   5 -- [100] >
```

The coordinates are defined in encoder counts for each axis.

The value displayed in brackets is the *value last recorded* for this position. If coordinate values have not yet been recorded for this position, the bracket is empty.

Press <Enter> to accept the displayed value, or enter a new value.

If the position requested is not valid, the coordinate values are not accepted, and an error message is displayed.

Examples: ■

```
DEFP PS
HERE PS
SETPV PS
```

Defines and records position PS. Then permits the user to reset the joint values for each of the axes.

Note: TEACH *pos* is the comparable command for recording an absolute XYZ position according to user defined values.

SETPV *pos axis var*

Description: Used for position modification, this command permits you to change one of the joint values of a previously recorded position.

The value of the coordinate which is modified by this command is defined in encoder counts.

SETPV *pos axis value* will not warn you of an invalid point coordinate until it tries and fails to reach it.

■ `SETPV PQ 1 500`	Used in a program, resets the joint value of the base axis by 500 encoder counts.

Notes: SETPVC *pos coord value* is the comparable command for changing the value of a Cartesian coordinate.

SETPVC

Format: `SETPVC` *`pos coord var`*

Where: *pos* is a recorded robot (group A) position,
coord is a Cartesian coordinate: X, Y, Z, P or R
: *var* is a constant or a variable expressed in tenths of a millimeter (X,Y,Z) or degrees (P,R)

Description: Used for position modification, this command enables you to change one of the Cartesian coordinates of a recorded position.

SETPVC will not warn you of an invalid point coordinate until it tries and fails to reach it.

Example: ■

```
SETP    PA=POSITION
SETPVC  PA X 250
SETPVC  PA Z ZV
SETPVC  PA P -90
```

The current coordinates of the robot are recorded for position PA, and then the position is modified by 25mm on the X-axis and -90° on the pitch axis; the Z-axis is modified according to the value of variable ZV.

Note: SETPV *pos axis var* is the comparable command for changing the value of a joint coordinate.

SHIFT / SHIFTC DIRECT/EDIT

Format: `SHIFT` *`pos`* `BY` *`axis var`*

Where: *pos* is a recorded position;
axis is an axis number;
var is a variable or a constant.

`SHIFTC` *`pos`* `BY` *`coord var`*

Where: *pos* is a recorded robot (group A) position;
coord is a Cartesian coordinate: X, Y, Z, P or R;
: *var* is a constant or a variable expressed in tenths of a millimeter (X,Y,Z) or degrees (P,R)

Description: Used for position modification, this command enables you to change the coordinates of a recorded position by an *offset value*.

`SHIFT`	Modifies joint coordinates; shifts the position by one joint value.
`SHIFTC`	Modifies Cartesian coordinates; shifts the position by one Cartesian value.

Examples:

■ `SHIFTC OBJ1 BY Z 500`	Shifts position OBJ1 up (Z-axis) by 50mm.
■ `SHIFT OBJ1 BY 3 1000`	Shifts position OBJ1 by 1000 encoder counts on axis 3.

Format:

```
SHOW DIN
SHOW DOUT
SHOW ENCO
SHOW DAC n
SHOW PAR n
SHOW SPEED
```

SHOW DIN

Description: Displays the status of the 16 individual inputs. 1 indicates ON; 0 indicates OFF.

If the command LSON has been activated, SHOW DIN will display the status of the axes’ home switches. The command LSOFF (default mode) resumes the display of input status.

Example: ■

```
>SHOW DIN
   1 -> 16: 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0
   O.K.
```

SHOW DOUT

Description: Displays the status of the 16 individual outputs. 1 indicates ON; 0 indicates OFF.

Example: ■

```
>SHOW DOUT
   1 -> 16: 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0
   O.K.
```

SHOW ENCO

Description: Displays the value of all encoders every 0.5 seconds.

The updated values will continue to flash until <Ctrl>+C is pressed.

Example: ■

```
>SHOW ENCO

   enc1 enc2 enc3 enc4 enc5 enc6 enc7 enc8
   1000 1000 1000 2371 2371 100  100  1000
```

SHOW DAC *n*

Description: Displays the DAC value for the axis *n* in millivolts.

Example: ■

```
>SHOW DAC 7
   DAC 7=0
   O.K.
```

SHOW PAR *n*

Description: Displays the value of system parameter *n*.

Example: ■

```
>SHOW PAR 261
   PAR 261=10
   O.K.
```

SHOW SPEED

Description: Shows the current speed settings.

Example: ■

```
>SHOW SPEED
   GROUP A SPEED IS: 40
   GROUP B SPEED IS: 50
   O.K.
```

SPEED

Format:

```
SPEED[A/B] var
SPEEDC var axis
```

Description: Sets the current speed value.

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

The speed is defined as a percentage. 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.

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

Examples:

■ `SPEED 60`	Sets movement of all axes to 60.
■ `SPEEDA 10`	Sets movement of all group A axes to 10.

Notes: The minimum and maximum speed values are determined by parameters 298 and 299.

Manual mode uses very slow movements whose speeds are relative to current speed settings. Parameter 297 determines the manual speed setting.

Refer to the SHOW SPEED command.

STAT DIRECT

Format: STAT

Description: Displays the status of active user programs. The list includes program priority and operation status.

A program will report as “PENDing” if it is waiting for a movement command to be completed.

Example: ■

```
>STAT
job name    priority    status
BOOM        000005      DELAY
DEMO        000005      PEND
OC  000005  DELAY
MVC 000005  PEND
SL          000005      SUSPENDED
```

EDIT

STOP

Format: `STOP [`*`prog`*`]`

Description:

`STOP` Aborts all programs, but movement commands remaining in the buffer continue and complete execution.

`STOP` *`prog`* Aborts the running of the specific program only.

Examples:

■ `STOP DEMO` Aborts program DEMO.

■ `STOP MYPRG` Aborts program MYPRG;
`CLRBUF` Clears the movement buffers, thereby halting all movements.

Notes: Refer to the CLRBUF command.

SUSPEND DIRECT/EDIT

Format: `SUSPEND` *`prog`*

Description: Suspends execution of the specified program.

The program completes the current movement command and all movement commands remaining in movement buffer, and then goes into suspension.

To resume execution of a suspended program from the point of suspension, use the CONTINUE command.

Example: ■

```
SUSPEND DEMO
```

Note: Refer to the CONTINUE command.

TEACH

Format: `TEACH` *`pos`*

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

Description: Records an absolute XYZ position, according to user defined values.

You are prompted to provide values for each of the Cartesian coordinates of the specified position in the following format:

```
>TEACH PP
   X --[200] >
   Y --[0] >
   Z --[340] >
   P --[-900] >
   R --[0] >
```

X, Y and Z values are in tenths of a millimeter, while pitch (P) and roll (R) values are in tenths of a degree.

The value in brackets is the *value last recorded* for this position.
Press <Enter> to accept the displayed value. If coordinate values have not yet been recorded for this position, the bracket is empty [.] .

If the position entered is not valid, the coordinates are not accepted, and the following message is displayed.

Note: SETPV *pos* is the comparable command for recording an absolute joint position according to user defined values.

TEACHR DIRECT

Format: `TEACHR` *`pos2`* `[`*`pos1`*`]`

Description: Where: *pos1* is a recorded robot (group A) position;
pos2 is defined for the robot (group A).

Description: TEACHR allows you to record a robot position relative to another position, or relative to the current position of the robot.

`TEACHR` *`pos2`*	Records the offset values of *pos2*, relative to the current position of the robot, in Cartesian coordinates. You must enter the offset values, as shown in the example below. *Pos2* will always be relative to the current position.
`TEACHR` *`pos2 pos1`*	Records the offset values of *pos2*, relative to *pos1*, in Cartesian coordinates. *Pos1* must be recorded before this command can be entered. *Pos2* will always be relative to *pos1*, moving along with and maintaining its offset whenever *pos1* is moved.

You are prompted to provide relative values for each coordinate of the specified position, as shown in the examples below.

X, Y and Z values are in tenths of a millimeter, while pitch (P) and roll (R) values are in tenths of a degree.

The value displayed in brackets is the *offset value last recorded* for this position. If coordinate values have not yet been recorded for this position, the bracket is empty [.] .

Press <Enter> to accept the displayed value, or enter a new offset value .

Examples:

-
  ```
  >DEFP OVER
  >TEACHR OVER
     X [.] > 0
     Y [.] > 0
     Z [.] > 1000
     P [.] > 0
     R [.] > 0
  ```

  Relative position OVER will always be 100mm vertically above the current position of the robot.

-
  ```
  >DEFP TABLE
  >DEFP OVER
  >HERE TABLE
  >TEACHR OVER TABLE
     X [0] >
     Y [0] >
     Z [0] > 2000
     P [0] >
     R [0] >
  ```

  Positions TABLE and OVER are defined.

  Current coordinates of robot are recorded for position TABLE;
  Relative position OVER is recorded as 200mm vertically above position PLACE. Whenever the coordinates of PLACE are changed, OVER will maintain a 200mm vertical offset.

TEST DIRECT

Format: TEST

Description: This command activates an internal system diagnostic procedure which checks the movement of the robot axes and the input/output functions of the controller. Alternately, the system can check the state of the homing microswitches instead of the inputs.

The TEST procedure is as follows:

- The system attempts to move each of the configured **axes** briefly in both directions. The axes are checked in sequence, beginning with axis 1. Each axis check results in an axis movement or in the message:

  ```
  TEST FAILURE AXIS n
  ```

  This message also appears when a defined axis does not actually exist.

- Upon completing the axis test, the system turns on all **outputs**, and then turns them off.

- The system then scans all the **inputs**. For each input which is on, the system turns on the corresponding output.

  The user should now short each input, in sequence, to ground; the system immediately response by turning on the corresponding output. For example, if input 7 is shorted, output 7 is turned on.

  To check the home microswitches (**limit switches**) instead of the inputs, enter the command LSON before executing TEST.

The TEST procedure is in the RUN state until it is aborted by means of the command A.

Notes: To activate TEST from the teach pendant, key in:

```
[RUN] 999 [ENTER]
```

Refer to the LSON command.

TON / TOFF

Format:

```
TON [n]
TOFF [n]
```

Description:

`TON`	Switches ON the thermic motor protection for all axes, or for specific axis.
`TOFF`	Switches OFF the thermic motor protection for all axes, or for specific axis.

The system will prompt you to confirm TOFF.

By default, the axes are in TON mode.

Note: *Warning! Use caution when in the TOFF mode; the motors are not protected by any software safeguards.*

TRIGGER EDIT

Format: `TRIGGER` *`prog`* `BY {IN/OUT}` *`n`* `[0/1]`

Where: *prog* is a program;
IN is an input; OUT is an output;
n is the I/O index, 1 ≤ *n* ≤ 16
0=off; 1=on.

Description: The TRIGGER command starts the execution of a specific program when the specified input or output is turned either on or off.

If *state* is omitted, execution of the program begins as soon as the specified I/O changes its state.

TRIGGER is a one-shot command. It execute a program only once, regardless of subsequent changes in the I/O state. You must repeat the TRIGGER command to reactivate the program it calls.

When used in the robotic system, sensors are connected to the controller inputs. The TRIGGER command enables the system to respond immediately and automatically to sensory signals whose timing is undefined or unpredictable. If such an application requires repeated sensor interrupts, the TRIGGER command must be entered prior to each expected sensor interrupt. The TRIGGER command can be included at the end of the called subroutine.

Examples:

■

```
TRIGGER WW BY OUT 8
```

Program WW is activated when output 8 changes its state.

■

```
   PROGRAM DRILL
   ****************
MOVE P28
SET OUT[3]=1
DELAY 500
SET OUT[3]=0
MOVE P27
END

   PROGRAM START
   ****************
TRIGGER DRILL BY IN 15 1
```

Program START activates program DRILL for the first time; thereafter, the TRIGGER command within program DRILL reactivates program DRILL whenever input 15 is turned on.

UNDEF

Format:

`UNDEF` *`pos`*

`UNDEF` *`pvect`*

`UNDEF` *`pvect`*`[`*`n`*`]`

Where: *n* is the index of a position in the vector.

Description:

Erases position values. The position is still defined, but does not have coordinate values.

`UNDEF` *`pos`*	Clears the coordinate values of the specified position.
`UNDEF` *`pvect`*	Clears the coordinate values of all the positions in the vector.
`UNDEF` *`pvect`*`[`*`n`*`]`	Clears the values of position *n* in the vector.

This command is useful when you intend to issue the APPEND command, since APPEND can assign coordinate values to a defined position only when it does not already values.

Examples:

■ `UNDEF VECTV[5]`	Clears the value of position 5 in vector VECTV.
■ `UNDEF VECTV`	Clears the values of all positions in vector VECTV.

Note:

This command does not create a program line.

VER DIRECT

Format: VER

Description: Displays the EPROM version and creation date.

Example: ■

```
>VER
   — ESHED ROBOTEC —
   VERSION: 1.42
   DATE : 21/07/91
```

WAIT

Format: `WAIT` *`var1 oper var2`*

Where: *var1* is a variable;
var2 is a variable or a constant;
oper may 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.

Example:

■ `WAIT IN[J]=1`	Waits until Input J is ON.
■ `WAIT X<Y`	Wait until the value of X is less than the value of Y.

* EDIT

Format: `*`*`user comment`*

Where *user comment* is a string of up to 40 characters and spaces.

Description: Allows you to annotate your programs.

The * character precedes textual comments within your program. These comments are not displayed during program execution.

Example:

■
```
*THIS IS AN EXAMPLE OF A COMMENT
```

@

Format: *@ directcom*

Where: *directcom* is a string written in DIRECT command format.

Description: Allows the execution of a DIRECT command from a running user program.

The @ commands relays the string to the controller as if it were a command entered in the DIRECT mode. However, the running program will not wait for the @ command to be executed. To make sure the command is executed before the program continues, enter a short delay command after each @ command.

Examples:

-
  ```
  @ SHOW DIN
  ```
  When program reaches this command line, the states of all inputs will be displayed.

-
  ```
  @ ATTACH LOAD
  DELAY 10
  @ LISTPV POSITION
  DELAY 10
  ```
  The DELAY command ensures the ATTACH command will be executed before the LISTPV command.

~ (Manual Control) DIRECT

Format:

```
~
<Alt>+M
```

Description: Activates and deactivates manual control of the robot from the keyboard.

When you press ~, Manual Keyboard mode is activated, and the following message is displayed:

```
MANUAL MODE!          or          MANUAL MODE!
>_                                >_
JOINT MODE                        XYZ MODE
```

The system’s response indicates the currently active coordinate system.

When you again press ~, Manual Keyboard mode is deactivated, and the following message is displayed:

```
EXIT manual mode
>_
```

When using **ATS**, if your keyboard does not include the ~ character, you can also toggle Manual Keyboard mode by pressing <Alt>+M.

Manual Keyboard mode enables several direct control operations from the keyboard, as described in the items.

Coordinate System

Manual Keyboard mode permits direct user manipulation of the axes:

Manual keyboard control varies, depending upon the currently active coordinate system. When in JOINT mode, the movement of individual axes is controlled; when in XYZ mode, the movement of the gripper tip is controlled.

When Manual Keyboard mode is active, use the following keys to change the movement coordinate systems:

`J`	Joint coordinate system
`X`	Cartesian (XYZ) coordinate system.

The following chart summarized the resulting movements when various keys are pressed. The axes will move as long as the activating key is depressed, or until a fixed stop is reached. The gripper will either open completely, or close completely.

KEY	JOINT mode	XYZ Mode
1 / Q	Move BASE (Axis 1) counterclockwise and clockwise.	All/some axes move in order to move the TCP (gripper tip) along the X+ and X- axis.
2 / W	Move SHOULDER (Axis 2) up and down.	All/some axes move in order to move the TCP (gripper tip) along the Y+ and Y- axis.
3 / E	Move ELBOW (Axis 3) up and down.	All/some axes move in order to move the TCP (gripper tip) along the Z+ and Z- axis.
4 / R	Move wrist PITCH (Axis 4) up and down	Shoulder, elbow and pitch axes move, causing the pitch angle to change while maintaining the position of the TCP (gripper tip).
5 / T	Move wrist ROLL (Axis 5) clockwise and counterclockwise (as seen from above, when gripper pointed down)	
6 / Y	Open and close electric gripper.	
7 / U	Move axis 7	
8 / I	Move axis 8	
9 / O	Move axis 9	
0 / P	Move axis 10	
– / [	Move axis 11	

Servo Control

When Manual Keyboard mode is active, you can switch servo control of the axes on and off. The following commands enable and disable control of all axes which are connected to the controller.

`C`	Turns on servo control of axes (CON).
`F`	Turns off servo control of axes (COFF).

If Manual Keyboard mode is not active, you can use the **ACL** commands CON and COFF for more specific activation of the axes.

Speed

When Manual Keyboard mode is active, you can set the speed of movement of all axes.

`S`	Sets speed of axis movement (SPEED).

You are prompted for a speed value—a percentage of the maximum speed.

```
SPEED.._
```

Type a number between 1–100 and press <Enter>

If Manual Keyboard mode is not active, you can use the **ACL** command SPEED for more specific speed definitions.

CHAPTER 4

Predefined System Elements

In addition to user commands and data elements, **ACL** has a number of predefined system elements which are used during the programming and operation of the robotic system.

Internal System Procedures

HOME

The HOME procedure performs a microswitch home search on all robot axes.

This procedure is activated either by entering the **ACL** command HOME, or by keying in RUN 0 from the teach pendant.

If the Robot Type is defined as 0 in the controller configuration, you must use command HOME *n* or HHOME *n* for each individual axis. Axes in group B and and group C must also be homed individually.

Refer to the command HOME in Chapter 3.

TEST

The TEST procedure performs a hardware diagnostic routine.

This procedure is activated either by entering the **ACL** command TEST, or by keying in RUN 999 from the teach pendant.

Refer to the command TEST in Chapter 3.

Reserved Program Names

ACL for **Controller-A** has two reserved names for user programs: AUTO and CRASH. Create and edit these programs in EDIT mode, like any other **ACL** user program. The system will run the program automatically, if it exists, when certain conditions occur:

AUTO

The AUTO program is automatically executed when the controller is powered on. The following items are suggested for inclusion in the AUTO program:

- I/O settings.
- ATTACH positions for teach pendant.
- RUN (execution of) user programs

Example

```
    PROGRAM AUTO
    *****************
HOME
DIMP PV
@ATTACH PV
DELAY 10
RUN OPER
END
```

When system is powered on or reset, the following occurs: the robot searches for its home position; a position vector PV is defined and attached to the teach pendant; program OPER is executed.

CRASH

The CRASH program is automatically executed when an impact, thermic, or “too large speed” error occurs. The following items are suggested for inclusion in the CRASH program:

- Commands to save the status of the system at the time of the crash.
- Messages to be sent to the host computer via the RS232 channel.

Example

```
    PROGRAM CRASH
    *****************
* OUTPUT 16 = EMERGENCY BUZZER
SET OUT[16]=1
PRINTLN "ROBOT HAS STOPPED"
PRINTLN "CHECK AND CORRECT PROBLEM"
PRINTLN "RESTART APPLICATION"
END
```

Position POSITION

POSITION is a system defined position, reserved for the coordinate values of the robot’s current position (location).

POSITION can be used for reading the values of the robot’s current position, and for assigning those values to variables or other positions.

Examples

Following are examples of commands which access and utilize POSITION.

-
  ```
  LISTPV POSITION
  ```

  Displays the currrent coordinate values of the robot arm.

-
  ```
  SETP 100=POSITION
  ```

  Position 100 receives the coordinate values of the robot’s current position. The equivalent of the command HERE 100 .

-
  ```
  SET var=PVAL POSITION 3
  ```

  Var receives the joint coordinate values of the specified axis (elbow) according to the robot’s current position.

-
  ```
  SET var=PVALC POSITION X
  SET var=PVALC POSITION Y
  SET var=PVALC POSITION Z
  SET var=PVALC POSITION P
  SET var=PVALC POSITION R
  ```

  Var receives the specified Cartesian coordinate value of the robot’s current position.

You can change the actual location of the robot by using POSITION, as shown in the following four examples.

Warning! The robot will immediately move to the new POSITION; therefore, make only small changes in the coordinates.

-
  ```
  SHIFT POSITION BY 2 100
  ```

-
  ```
  SHIFTC POSITION BY Z 100
  ```

-
  ```
  SETPV POSITION 1 500
  ```

-
  ```
  SETPVC POSITION Y 300
  ```

System Variables

System defined variables contain values which indicate the status of inputs, outputs, encoders, and other control system elements. The **ACL** system variables enable you to perform diagnostic tests and recovery programs, and to execute applications which require real-time information about the system's status.

ACL for **Controller-A** contains 9 system variables:

```
IN[16]
OUT[16]
ENC[11]
TIME
LTA
LTB
MFLAG
ERROR
ANOUT[11]
```

The indices indicate the dimensions of the array variables.

The values of the system variables are manipulated in the same manner as user defined variables. However, system variables cannot be deleted, and some are read-only.

The values of variables IN, ENC, LTA, LTB, TIME, MFLAG are updated continuously. Since any value assigned to these variables will be overwritten immediately, they are considered read-only variables.

IN[*n*]

The value of this variable indicates the state of the specified input.

The value of IN[*n*] is updated at each controller clock tick according to the actual state of the input. Any value written to this variable will be overwritten within one clock tick.

IN[*n*] is considered a read-only variable.

n = the index of the input; may be a variable or a constant; may not exceed the number of inputs configured.

Examples

Purpose	Program Command	Display	Notes
To view the current status of the input.	`PRINTLN IN[3]`	`1`	=input is ON
		`0`	=input is OFF

To control programs running in a work cell.	`IF IN[I]=0`
	`  SET  OUT[2]=1`
	`ENDIF`		

OUT[*n*]

The value of this variable determines the state of the specified output.

The value of OUT[*n*] is applied to the actual output at each controller clock tick.

OUT[*n*] is a read/write variable.

n = the index of the output; may be a variable or a constant; may not exceed the number of outputs configured.

Examples

Purpose	Program Command	Display	Notes
To view the current status of the input.	`PRINTLN OUT[7]`	`1`	=output 7 is ON
		`0`	=output 7 is OFF

To check and change status of device connected to an output.	`IF OUT[5]=0`
	`  SET OUT[5]=1`
	`ENDIF`
			If output 5 (e.g., lamp) is off; turn it on.

To change the state of an output.	`SET OUT[5]=1-OUT[5]`		

ENC[*n*]

The value of this variable indicates the number of encoder counts for the specified axis at its current position.

The value of ENC[*n*] is updated at each controller clock tick according to the actual state of the encoder. Any value written to this variable will be overwritten within one clock tick.

ENC[*n*] is considered a read-only variable.

n = the index of the axis; may be a variable or a constant; may not exceed the number of axes configured.

Examples

Purpose	Program Command	Display	Notes
To view the current status of the input.	`PRINTLN ENC[1]`	`0`	ENC[1]=0 encoder counts.
		`6844`	ENC[1]=6844 encoder counts.
To assign the encoder value to a variable.	`SET X=ENC[5]`		The value of encoder 5 is written to X.

TIME

This variable contains the current value of the controller clock.
When the controller is turned on or reset, the clock is initialized to 0. Every

10 milliseconds (tick) the controller clock value is incremented by 1.

TIME is considered a read-only variable.

Example

Purpose	Program Command	Display
To determine the actual duration of the executed movement.	PROGRAM TIME **************** SET TIMEA=TIME MOVED POS99 SET TIMEA=TIME-TIMEA PRINTLN "MOVE DONE IN " PRINT TIMEA " MS"	MOVE DONE IN 500 MS

LTA and LTB

The values of these variables indicate the time (that is, controller clock value; as for TIME variable) at which the specified axis group will reach the target position last received.

LTA applies to axis group A. LTB applies to axis group B.

These variables are used when movements commands MOVE, MOVEC, and MOVES are placed in the buffer. These variables enable practical scheduling and work cell synchronization; for example: conveyor pick-up, synchronization of two axis groups, and so on.

LTA and LTB are considered read-only variables.

Example

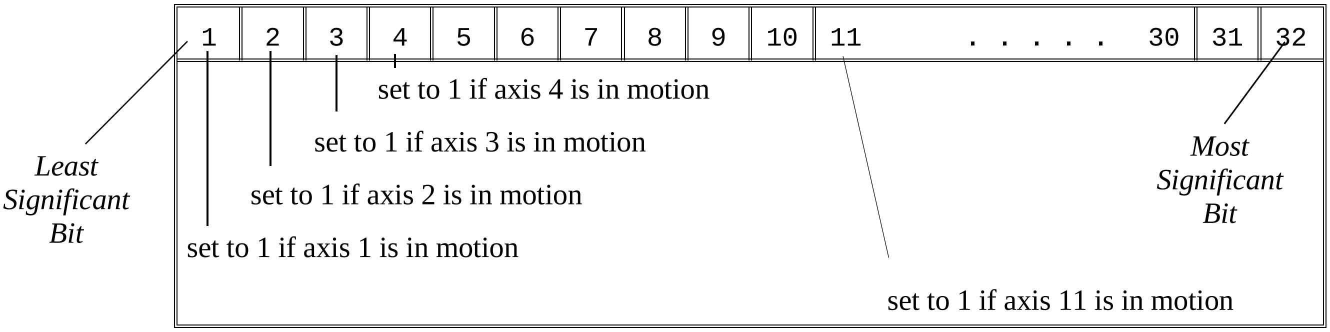
Purpose	Program Command
To synchronize the arrival of group A axes and group B axes at their respective destinations. (POSA5 is a position of group A and POSB3 is a position of group B.)	PROGRAM SYNCH ************** MOVE POSA5 SET V=LTA-TIME MOVE POSB3 V

MFLAG

The value of this variable indicates which axes are currently in motion.

MFLAG is considered a read-only variable.

Whenever a MOVE command is executed, the 32 bits of the binary representation of MFLAG are switched on, according to the following chart:


1 2 3 4 5 6 7 8 9 10 11 . . . . . 30 31 32
set to 1 if axis 4 is in motion
set to 1 if axis 3 is in motion
set to 1 if axis 2 is in motion
set to 1 if axis 1 is in motion
set to 1 if axis 11 is in motion
Least Significant Bit
Most Significant Bit

Bits 12 through 32 are always set to 0.

Assuming the controller is configured with five axes in group A, a servo gripper at axis 6, two axes in group B, and three axes in group C, the value of MFLAG will indicate movement of the axes as shown in the following chart:

Bit Value	1	2	4	8	16	32	64	128	256	512	1024
Group	**Group A**					**Gripper**	**Group B**		**Group C**		
Axis	1	2	3	4	5	6	7	8	9	10	11

Examples

Purpose	**Program Command**	**Display**	**Notes**
Movement status of the axes.	`PRINTLN MFLAG`	31	31=1+2+4+8+16; All axes in group A are currently in motion.
	`PRINTLN MFLAG`	543	543=31+512; All axes in group A and axis 10 are currently in motion.

ERROR

When a system error occurs during run time, the ERROR variable is assigned a value. The value of ERROR indicates the specific error. An error message is also displayed.

Refer to Chapter 5 for explanations of the run time error messages.

The ERROR variable *must have an initial value of 0*; otherwise, the value of the variable will not change during program execution.

This is a read/write variable.

Example

Purpose	Program Command	Display	Notes

To identify run-time errors.	`PROGRAM MOVE`
	`**************`
	`LABEL     1`
	`MOVE POS66`
	`MOVE POS67`
	`GO TO     1`
	
	`PROGRAM ERROR`
	`***************`
	`SET ERROR=0`
	`* wait for error to occur`
	`WAIT ERROR <>0`
	`PRINTLN ERROR`
		53	Simultaneously run programs ERROR and MOVE.
			53=error ID number

ANOUT[*n*]

This variable contains the DAC value currently being applied to the specified motor driver.

The value of ANOUT[*n*] is applied to the specified axis at each controller clock tick.

When SET ANOUT is executed, servo control of the axis is disabled; COFF is in effect until CON is activated for the axis.

ANOUT[*n*] is a read/write variable.

n = the index of the axis; may be a variable or a constant; may not exceed the number of axes configured.

The DAC value is in the range: –5000 to +5000.

Warning! Use with care to avoid motor damage.

Example

Purpose	Program Command	Display	Notes
To view the current DAC value.	`PRINTLN ANOUT[5]`	`2500`	DAC set to half of maximum value.
To set the DAC value of an axis.	`SET ANOUT[1]=1000`		Sets the analog output value for axis 1 to 1000.

CHAPTER 5

System Messages

Run Time Error Messages

When a system error occurs during run time, the system variable ERROR is assigned a value corresponding to the specific type of error, and an error message is displayed. Refer to the section “System Variables,” in Chapter 4.

Following is a description of the system errors you may encounter.

Errors in Arm Movement

53 `*** IMPACT PROTECTION axis` *n*

The controller has detected a position error which is too large. The system aborted all movements of that axis group, and disabled all axes of that group. The user routine CRASH, if it exists, has been executed. Possible causes:

(1) An obstacle prevented the movement of the arm.

(2) An axis driver fuse has blown.

(3) An encoder fault.

(4) A mechanical fault.

(5) The axis is not connected.

- Determine and correct the cause of the position error. Then reenable servo control of the motors (CON), and restart the program.

54 `*** OUT OF RANGE axis` *n*

An attempt was made to record a position (HERE) while the robot arm was out of its working envelope.

- Manually move the arm to a location within its working envelope. Then repeat the command.

55 `*** THERMIC OVERLOAD axis` *`n`*

Through a software simulation of motor temperature, the system has detected a dangerous condition for that motor. The system aborted all movements of that axis group, and disabled all axes of that group. The user routine CRASH, if it exists, has been executed. Possible causes:

(1) The arm attempted to reach a position, which could not be reached due to an obstacle (for example, a position defined as being above a table, but actually slightly below the table's surface). The impact protection is not activated because the obstacle is close to the target position. However, integral feedback will increase the motor current and the motor will overheat, subsequently causing the Thermic Protection to be activated.

(2) An axis driver is faulty or its fuse has blown.

(3) The robot arm is near to the target position, but does not succeed in reaching it, due to a driver fault. The software will then detect an abnormal situation.

(4) The Thermic Protection parameters are improperly set, or have been corrupted by improper loading of parameters.

- Check the positions, the axis driver card and parameters. Reenable servo control of the motors (CON).

Errors During Program Execution

302 `ARITHMETIC OVERFLOW AT LINE` *`n`*

The result of a mathematical operation is out of range (or invalid).

303 `NO POSITION ASSIGNED TO POINT` *`pos`*

The position has been defined using DEFP command, but its coordinates have not been recorded or set.

- Use HERE or other commands to assign coordinates to the position.

304 `AXIS DISABLED`

You have attempted to move an axis which is not enabled. Possible causes:

(1) A movement command could not be executed because servo control of the arm has been disabled (COFF).

(2) A previous movement of the arm resulted in an Impact or Trajectory error, thereby activating COFF and disabling the arm.

- Check the movements of the robot, and correct the command(s).

305 `TOO DEEP NESTING`

Too many GOSUB subroutines are nested within one another.

306 `INVALID PROGRAM`

The RUN, GOSUB, TRIGGER command cannot be executed, due to faulty syntax or logic in the program. For example, the program contains an IF command without a corresponding ENDIF command.

309 `INDEX OUT OF RANGE`

You have attempted to use an index value which is beyond the defined range of the variable array or position vector.

310 `BAD AXIS`

The axis is not in the group specified by the command, or the axis is not configured.

311 `LOOPING RELATIVE CHAIN OR DEPTH EXCEEDED 32 POINTS`

ACL permits relative positions to be linked to one another in a chain of up to 32 positions. This relative chain of positions must be anchored to one absolute (root) position.

You attempted to define a relative position. The error may be:

(1) One of the positions encountered in the relative chain is the position you attempted to record (a position cannot be relative to itself).

(2) You have linked the relative positions in an invalid or infinite loop.

(3) You have linked more than 31 relative positions.

312 `BAD POINT COORDINATE` *`pos`*

You attempted to use an invalid position. For example, the relative position you have defined is out of the axis’ range.

- Record new coordinates for the position.

315 `INCOMPATIBLE POINTS`

Possible causes:

(1) You have attempted to use the HERE command for positions in different axis groups, or positions which are both of group C but assigned to different axes.

(2) You attempted to copy a position (SETP) from one axis group to another axis group.

316 `NO GRIPPER CONFIGURATION`

You attempted to use a command which indicates the presence of a gripper. The command cannot be executed because a gripper has not been configured.

317 `BAD CARTESIAN POINT` *`pos`*

The position could not be recorded or reached because its XYZ coordinates are out of the XYZ envelope.

- Switch to JOINT mode.

CHAPTER 6

User Memory Configuration

The standard **Controller-A** has 128KB of battery-backed user RAM, which is allocated during the controller configuration.

For example, when the configuration is performed by means of the **ATS** hot-key combination **<Ctrl>+F1**, and the options SCORBOT VII and 8 AXES are selected, the default memory allocation for each type of element will be as follows:

150	Programs
3000	Program lines
600	Variables
2380	Group A positions
2380	Group B positions
0	Group C positions
550	Comments

Only axis control groups A and B are defined in the default controller configuration. Group A is defined as axes 1 through 5. Axis 6 is configured for an electric gripper. All remaining axes are defined as Group B. To define Group C axes, the **ACL** command CONFIG must be used.

The number of data elements are calculated according to the specific amount of memory required by each element, as follows:

Data Element	Memory
Program header	11 bytes
Program line	10 bytes
User variable	16 bytes
Group A or Group B position	[(*no. axes in the group* +1) × 2 + 8] bytes
Group C (single axis) position	14 bytes
User comment/string	12 bytes

System parameters and variables, together with delimeters, can occupy up to 2.5KB.

The sum of all elements must not exceed the controller’s available memory, or 128KB.

The system requires a minimum number of some data elements, as shown below. If you specify a number less than the minimum, the system will automatically assign that element the minimum required.

Data Element	Minimum
User programs	2
Program lines	50
User variables	10
User string/comments	50

Example:

The controller is configured for 11 axes:

Group A is defined as 5 axes — axes 1 through 5 (the robot).
The gripper is defined as axis 6.
Group B is defined as 4 axes — axes 7 through 10.
Group C is thus left with one axis only — axis 11.

In addition, the following data elements are defined:

100 Programs	= 100 × 11 = 1100 bytes
550 Lines	= 550 × 10 = 5500 bytes
200 Variables	= 200 × 16 = 3200 bytes
450 Positions for group A	= 450 × 20 = 9000 bytes
450 Positions for group B	= 450 × 18 = 8100 bytes
150 Positions for group C	= 150 × 14 = 2100 bytes
100 Strings/comments	= 100 × 12 = 1200 bytes
Total user BBRAM	30200 bytes
BBRAM reserved for system	2560 bytes
Total BBRAM	32760 bytes

The total user memory needed for this configuration is 30200. Together with the maximum allocation for system memory of 2560 bytes, a total of 32760 bytes is required. The configuration is valid because 32760 < 131072; (131072=128×1024).

(This example is also applicable and valid for controllers equipped with only 32KB of BBRAM; 32760 <32768 ; (32768 = 32×1024).)

CHAPTER 7

Parameters

Many of the controller functions depend on the setting of the system parameters. System parameters determine operations and conditions such as:

- Servo control
- Work envelope
- Axis protection
- Speed limits
- Gripper operation
- Teach pendant and manual operation
- Cartesian kinematic calculations

Warnings

- Only skilled operators should attempt to manipulate parameters.
- Backup your current system parameters before you change parameter values.
- Activate COFF before you change parameter values.
  Never change parameter values while robot is in motion.
  Never change parameters values while programs are running.
- Be sure impact protection parameters are properly set. These parameters monitor the servo axes for abnormal conditions, such as encoder and power failure, and impact. When such conditions are detected, the motors are disabled. Working without active impact protection may result in damage to the robot arm.

Accessing Parameters

The parameters in the **Controller-A** may be accessed by the following user commands:

`SHOW PAR` *`n`*
DIRECT mode.
Displays the value of parameter *n*.

`LET PAR` *`n`*`=`*`var`*
DIRECT/EDIT modes.
Changes the value of parameter *n* to *var*
(either a constant or a variable).

`SET` *`var1`*`=PAR` *`n`*
DIRECT/EDIT modes.
Assigns the value of parameter *n* to a variable.

`SENDPAR`
DIRECT mode.
Generates a listing of all system parameters, which, if captured into a file, can later be transmitted to the host computer by means of the RECEIVE and APPEND commands.

`INIT CONTROL`
DIRECT mode.
Must be issued after changing a parameter; otherwise the new value will not take effect.

`INIT PROFILE`
DIRECT mode.
Must be issued after changing parameter 76; otherwise the new value will not take effect.

To view a current parameter value, use the command SHOW PAR

For example: `SHOW PAR 21 <Enter>`

To change a parameter value, use the command LET PAR

For example: `LET PAR 21 200 <Enter>`

You must then issue the command INIT CONTROL.

Parameter Descriptions

Controller-A has two types of parameters:

- Parameters applicable to a device regardless of the axis to which it is connected. For example, PAR 176 defines the DAC value appplied to the gripper motor at the start of gripper movement.
- Parameters which are applied to each axis individually. These parameters are allotted a range of numbers, at intervals of 20, in the controller's table of parameters. The range is indicated by the term PAR *n+axis*; for example PAR 180+*axis*.

  Parameters 23, 43 and 63, for example, are servo control parameters for axis 3; parameters 69, 70 and 71 define the integral feedback constants for axes 9, 10, and 11, respectively.

Parameters which define DAC (analog output) values are in the range ±5000 (equivalent to ± 24V on motor). Other parameter values are in units such as encoder counts, controller clock ticks, linear measurements, and so on.

The parameters supplied with the **SCORBOT-ER Vplus** and **SCORBOT-ER VII** are appropriate for most robotic applications.
Do not change them unless necessary.

Some parameters are only valid for a specific robot configuration. The effect of others may change depending on the robot arm used. Read the documentation carefully before making parameter changes.

Some parameters are only operational in EPROM version 1.32 and later, or version F.44 and later. Check the EPROM version (by means of VER command) before attempting to use these parameters.

The controller for **SCORBOT-ER VII** must be equipped with EPROM version F.44 or later, to permit use of a gripper with encoder feedback.

Axis Servo Control Parameters

These parameters determine the servo control loop for each axis individually.

`PAR 20+`*`axis`*

Defines the value of the proportional feedback constant.

`PAR 40+`*`axis`*

Defines the value of the differential feedback constant.

`PAR 60+`*`axis`*

Defines the value of the integral feedback constant.

`PAR 80+`*`axis`*

Defines the offset for DAC output; the absolute value increases the DAC feedback result. Sets the minimum DAC output when the axis is under servo control.

Range: 0 – 5000.
Typical value: 0 – 200.

Global Servo Control Parameters

These parameters determine the scaling of servo feedback constants when the arm is moving or completing a movement.

`PAR 78`

If PAR 78 ≠ 0, the proportional and differential feedback constants are doubled at the end of motion.

`PAR 79`

Defines the ratio of reduced integral feedback constants while axes are in motion. During movement the integral feedback constants are divided by PAR 79.

Axis Limits Parameters

These parameters determine the limits of axis motion, in encoder counts.

`PAR 100+`*`axis`*

Upper limit of axis movement.

`PAR 120+`*`axis`*

Lower limit of axis movement.

Axis Position Error Parameters

`PAR 260+`*`axis`*

Defines the maximum position error, in encoder counts, allowed for the completion of a MOVED, MOVELD or MOVECD command. This parameter is applied only when the EXACT mode is active.

If the axis is a servo gripper, PAR 260+*axis* defines the maximum fluctuation of the encoder value while the gripper blocked.

To use PAR 260+*axis* for a servo gripper on a **SCORBOT-ER VII**, the controller must be equipeed with EPROM version F.44 or later.

Smoothing Parameter

`PAR 77`

Defines a smoothing factor for linear and circular motions.
If PAR 77 > 0, points (coordinate values) along the linear/circular path are averaged in order to reduce arm vibration.

The smoothing is increased if PAR 77 is increased.

Range: 0 – 4

If the parameter is too high the arm may jump slightly at the end of the motion

Valid only for EPROM version 1.32 and later.

Velocity Profile Parameter

`PAR 76`

Determines the speed profiles for robot movements. The parameter value determines the minimum velocity during acceleration or deceleration. Higher values increase the acceleration/deceleration speed.

Typical values: 0–10

The command INIT PROFILE must be executed following any changes in the value of PAR 76.

Thermic Protection Parameters

Thermic protection turns off motor power if a motor is stalled for an abnormal length of time. These parameters are used to define and calculate the thermic error threshold for each axis.

`PAR 140+`*`axis`*

Motor voltage/speed characteristics of a free running motor, used to calculate the average thermic load of the motor, defined by:

$$r = \frac{\textit{logical DAC output} \times 65.53}{\textit{number of encoder counts per clock tick}}$$

One clock tick is 10ms. Assuming DAC=5000:

$$r = \frac{327650}{\textit{number of encoder counts per } 10\textit{ms}}$$

At 5000 DAC, a standard **SCORBOT-ER Vplus** motor with a 20-slot encoder has a speed of 8555 RPM, resulting in 32 counts per tick, and thus giving a typical motor characteristic value of 3000.

The current analog output value combined with the motor characteristic value result in a theoretical thermic analog output value, which is equivalent to the amount of power not translated into movement.

`PAR 160+`*`axis`*

Defines the thermic error threshold; that is, the maximum DAC value that can be applied to a stalled motor.

When the theoretical thermic analog output value exceeds the value of PAR 160+*axis*, a thermic error is announced, and power to a stalled motor is shut off.

Range: 1000 – 5000
Typical value: 4000

Impact Protection Parameters

These parameters define the impact error conditions for each axis.

`PAR 180+`*`axis`*

Defines the number of times to check for actual motor blockage before announcing an impact condition.

`PAR 240`

Defines the axis group which responds when an impact condition is detected:
If PAR 240=0: all motors are stopped (default).
If PAR 240 ≠ 0: only motors in group to which impacted motor belongs are shut off.

Par 240 is valid only for EPROM version 1.42 and later.

`PAR 240+`*`axis`*

Defines the minimum number of encoder counts per tick while axis is supplied with the power specified in PAR 280+*axis*.

If the DAC output value is greater than specified in PAR 280+*axis* and the amount of movement is less than that specified in PAR 240+*axis*, an impact is suspected.

If PAR 240+*axis* = 0: Impact protection not applied.

Warning! Working without an active impact protection may damage the robot arm.

`PAR 280+`*`axis`*

Defines the DAC output value above which the impact algorithm starts checking for motion impact.

Value should usually be greater than 4000.

Speed Limit Parameters

These parameters define the effective robot speeds in encoder counts per clock tick. (Clock tick = 10 ms). Values for PAR 297, PAR 298 and PAR 299 are **(***encoder counts / clock tick*) × 1000.

`PAR 298`

Minimum speed.

`PAR 299`

Maximum speed.

`PAR 297`

Defines the transition speed threshold, below which the very low speed manual movement algorithm is applied.

The low speed algorithm accumulates movement requests, resulting in an improved maneuverabilty.

The regular algorithm does not accumulate manual movement requests, thereby avoiding high speed overshoots .

`PAR 294`

Defines the maximum manual speed of Cartesian motions.
Value is: (0.1 mm / clock tick) × 1000.
Typical value for **ER Vplus** : 8000
Typical value for **ER VII** : 13000

`PAR 296`

A speed decreasing factor for peripheral axes.
Value is a percentage of robot speed.
Par 296 is valid only for EPROM version 1.32 and later.

Manual Speed Parameters

`PAR 220+`*`axis`*

Defines the relative speed setting for each axis when the teach pendant is used to move the axes. Range differs from regular speed settings.

The value defines the relative speed among axes and not the actual physical speed.

Keyboard Manual Parameter

`PAR 300`

Defines the number of times a keyboard stroke needs to be repeated in order to produce a smooth continuous movement during manual keyboard operation of the axes.

Encoder Interface Parameters

`PAR 300+`*`axis`*

Determines a division relation between the actual encoder count and the encoder count that is used for **ACL** control. May be required because of the ±32767 encoder value limit.

The value of the parameter defines the number of times each encoder value is divided by 2. For example:

If PAR 300+*axis* = 1 : encoder count is divided by 2.

If PAR 300+*axis* = 2 : encoder count is divided by 4.

If PAR 300+*axis* = 3 : encoder count is divided by 8.

Gripper Parameters

These parameters determine the gripper operation.

`PAR 72`

Defines the amount of time it takes the gripper to close.
Value is in hundredths of a second.
Typical value: 125 (1.25 seconds).

PAR 72 is valid only for an electric gripper *without* encoder feedback. Also valid only for EPROM version 1.32 and later.

`PAR 73`

Defines the gripper encoder range; that is, the number of encoder counts it takes for the gripper to close from a fully opened position.

If PAR 73 = 0: no encoder on the gripper motor.

PAR 73 is valid only for a gripper with encoder feedback.

`PAR 74`

The value of the encoder count of the gripper at the closed position.

PAR 74 is valid only for a gripper with encoder feedback.

`PAR 75`

Defines the DAC value to be applied to the gripper motor when OPEN and CLOSE commands are executed.

Range: 0 – 5000.

When using a standard **SCORBOT-ER Vplus** gripper (equipped with encoder), do not set PAR 75 higher than 3500.

`PAR 275`

Defines the DAC value to be applied to the gripper motor after the completion of a CLOSE command.

This DAC value determines a constant holding power of the gripper. You must be careful not to set the value too high.

Typical value: 1000.

PAR 275 is valid only for an electric gripper *without* encoder feedback. Also valid only for EPROM version 1.32 and later.

`PAR 276`

Defines the DAC value to be applied to the gripper at the start of movement.

PAR 276 is valid in EPROM version F.44 and later.

`PAR 277`

Defines the duration of parameter 276.

Value is in hundreths of a second.

PAR 276 is valid in EPROM version F.44 and later.

Homing Parameters

`PAR 200`

If PAR 200 = 0 (default): a high precision, double homing routine is performed.
If PAR 200 = 1: a faster, less precise, homing routine is performed.

Par 200 is valid only for EPROM version 1.42 and later.

`PAR 200+`*`axis`*

Defines the maximum DAC value allowed while homing. If analog output reaches this value while homing, the homing routine will interpret it as a mechanically blocked motor and will change the direction of the search or stop the homing procedure.

5000 is for the maximum DAC voltage.

`PAR 212`

Used to slow motion during homing.

Defines the number of clock ticks for which movement is halted between each (feedback) interval while searching for limit switch.

If PAR 212 = 0 : (default) no slowing during homing.

Typical value: 1 or 2

Par 212 is valid only rsion 1.42 and later.

`PAR 295`

Defines the relative speed of movement when homing an individual axis using the HOME *axis* command.

The value of the parameter is a percentage of the encoder speed when homing axis 1; that is, a value of 100 equals the encoder homing speed of axis 1.

Note that homing speed is also determined by the encoder interface parameter, PAR 300+*axis*.

PAR 295 serves to coordinate the different encoders used in the robot and peripheral devices.

Cartesian Calculations Parameters

These parameters define the mechanical arm lengths, encoder and gear ratios, and the robot’s home position. These parameters are used to calculate the Cartesian position of a vertically articulated robot arm.

The coupling or decoupling of the axes is determined by the Robot Type specified in the controller configuration.

- **SCORBOT-ER Vplus** is a decoupled robot. In a decoupled robot, a shoulder movement (axis 2) does not change the world angle of the elbow (axis 3) and pitch axis. When the elbow joint moves, the pitch axis maintains its angular position relative to world.
- **SCORBOT-ER VII** is a coupled robot. In a coupled robot, all joints are coupled such that a movement in axis 2 changes the world angle of axis 3 but maintains the relative angle between joints 2 and 3. This relation applies to other axes as well.

Note that pitch and roll motions in the **SCORBOT-ER Vplus** are created by combining the rotation of motors 4 and 5. In the **SCORBOT-ER VII** separate motors movie each of these axes.

Rotation Scaling Parameters

`PAR 33`

Number of encoder counts for +90° rotation of axis 1.

`PAR 34`

Number of encoder counts for +90° rotation of axis 2.

`PAR 35`

Number of encoder counts for +90° rotation of axis 3.

`PAR 36`

- **ER Vplus**: Difference in number of encoder counts (encoder 4 – encoder 5) for +90° rotation of axis pitch.
- **ER VII**: Number of encoder counts for +90° rotation of axis 4.

`PAR 37`

- **ER Vplus**: Total number of encoder counts (encoder 4 + encoder 5) for +90° rotation of axis roll.
- **ER VII**: Number of encoder counts for +90° rotation of axis 5.

Horizontal Reference Position Parameters

These parameters define the encoder offset from the home position to a position in which all axes are aligned and in the horizontal position, including a horizontal gripper plane (position H). This position may also include an offset for axis 1.

`PAR 52`

Value of encoder 1 at position H.

Valid only for EPROM version 1.32 and later.

`PAR 53`

Value of encoder 2 at position H.

`PAR 54`

Value of encoder 3 at position H.

`PAR 55`

Value of encoder 4 at position H.

`PAR 56`

Value of encoder 5 at position H.

Valid only for **SCORBOT ER VII**.
Valid only for EPROM version 1.32 and later.

Length Parameters

The values of the length parameters are in tenths of millimeter.

`PAR 92`

Y coordinate (offset from center along the Y-axis) of the gripper tip when robot is in the home position.

Par 92 is valid only for EPROM version 1.4 and later.

`PAR 93`

X coordinate of the rotation axis of arm link 2 when the robot is in the home position.

`PAR 94`

Z coordinate of the rotation axis of arm link 2.

`PAR 95`

Length of the arm link from the first articulated joint.

`PAR 96`

Length of arm link from the second articulated joint.

`PAR 97`

Distance from pitch axis to the tip of the gripper.