ACL

Advanced Control Language

Version F2.28
For Controller-B

Reference Guide

Catalog #100085 Rev. A

intelitek

ACL Advanced Control Language

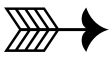
Catalog # 100085 Rev. A

February 1995

INTELITEK INC.
444 East Industrial Park Drive
Manchester NH 03109-537
Tel: (603) 625-8600
Fax: (603) 625-2137
Web site www.intelitek.com

Table of Contents

CHAPTER 3 The ACL Commands

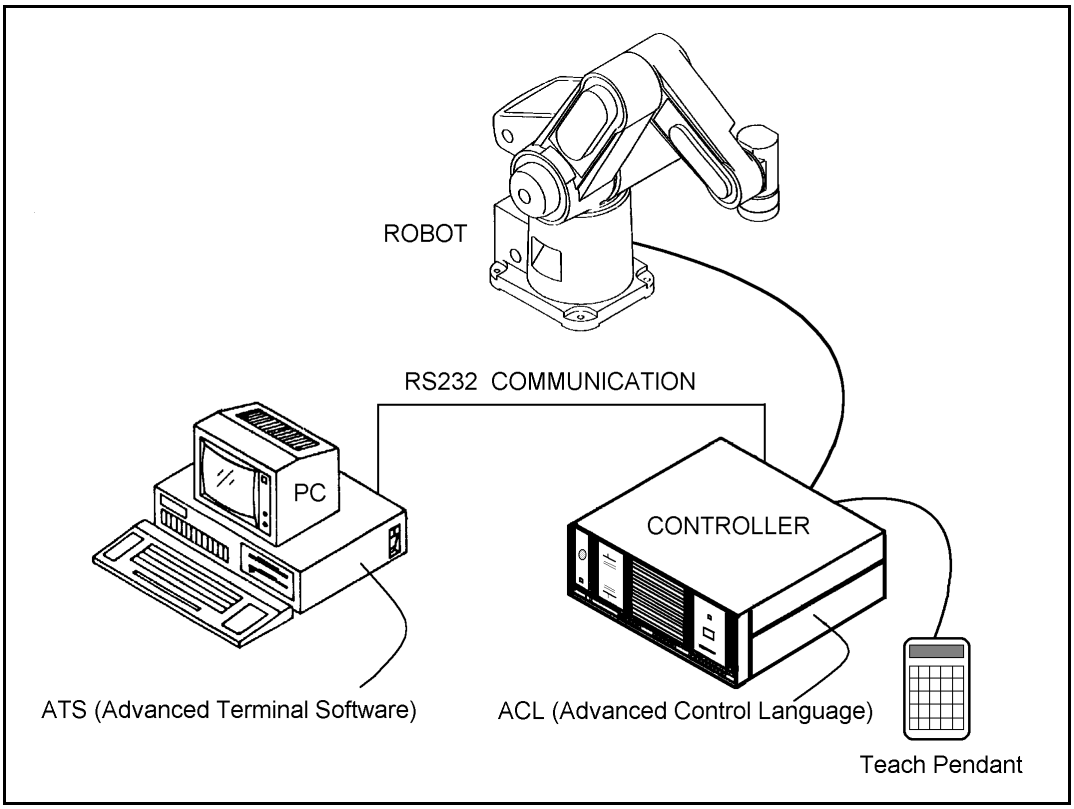


Introduction

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-B**, 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.



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.

This *Reference Guide* is a complete guide to **ACL** for **Controller-B**.

ATS features include the following:

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

ATS for **Controller-B** is the subject of a separate manual.

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.

Some commands are password-protected, and can be issued only when the PRIVILEGE mode is active.

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 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 699).

Refer to Chapter 7 for detailed descriptions of parameters.

Axis Control Commands

MOVE	SPEED	INT_ON
MOVED	SPEEDL	INT_OFF
MOVEC	SHOW SPEED	TON
MOVECD		TOFF
MOVEL	EXACT	
MOVELD	PROFILE	HOME
MOVES		
MOVESD	CON	CLR
SPLINE	COFF	ZSET
SPLINED		MODULO ROLL
	SET ANOUT	
OPEN	SHOW DAC	~
CLOSE		<Alt>+M
JAW		AUTO
CLRBUF		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 time 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.
SPLINE				
	`SPLINE` *`pvect pos1 posn`*	Moves axes smoothly through *or near* all consecutive vector positions between position 1 and position *n*. Constant speed betweeen consecutive positions. Joint SPLINE: Absolute Joint positions in vector; axes move at current joint speed. Linear SPLINE: Absolute XYZ positions in vector; robot's TCP moves at current linear speed.	DIRECT, EDIT	
	`SPLINED` *`pvect pos1 posn duration`*	Same as SPLINE, except average speed determined by time definition.	DIRECT, 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. $0 \le var \le 5000$
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. $0 \le var \le 5000$
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. $0 \le var \le 100$
	`JAW` *`var duration`*	Same as JAW, except speed determined by time definition.	DIRECT, EDIT	Use with caution. May damage motor. $0 \le var \le 100$

COMMAND FORMAT	DESCRIPTION	MODE	NOTES
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	
SPEED			
`SPEED` *`var`*	Sets speed for group A axes. Determines speed of MOVE, MOVES and Joint SPLINE movements.	DIRECT, EDIT	$1 \leq var \leq 100$. Default is 50.
`SPEED{A/B}` *`var`*	Sets speed for group A or group B.	DIRECT, EDIT	
`SPEEDC` *`var axis`*	Sets speed for axis in group C.	DIRECT, EDIT	
SPEEDL			
`SPEEDL` *`var`*	Sets speed for robot (group A) axes. Determines speed of MOVEL, MOVEC and Linear SPLINE movements.	DIRECT, EDIT	*var* = mm/second
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, SPLINED.	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	
PROFILE			
`PROFILE SINUS {A/B/C}`	Applies sinusoid profile to trajectory: fast acceleration and deceleration at start and end of movement, with constant speed along path.	DIRECT, EDIT	SINUS is default mode.
`PROFILE PARABOLE {A/B/C}`	Applies parabole profile to trajectory: slow acceleration until maximum speed is reached; deceleration at same rate.	DIRECT, EDIT	

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
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	
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 caution.* May damage motor.
SHOW DAC				
	SHOW DAC *axis*	Displays the value of DAC in millivolts.	DIRECT	1 ≤ *axis* ≤ 12
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	

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
CLR				
CLR *n*		Initializes (sets to 0) the value of a specific encoder.	DIRECT, PRIV	$1 \leq n \leq 12$
CLR *		Initializes (sets to 0) the value of all encoders.	DIRECT, PRIV	
ZSET				
ZSET		Initializes (sets to 0) the value of the index pulse on all encoders.	DIRECT	
MODULO ROLL				
MODULO ROLL		Returns the value of the roll axis to a value within ±360° range, without moving the axis.	DIRECT	
~				
~ or <Ctrl>+M		Activates and deactivates Manual mode, for direct control of axes from terminal or computer keyboard .	DIRECT	
AUTO				
AUTO		Transfers control to the keyboard after the Auto/Teach switch on the teach pendant is switched to Auto.	DIRECT	
TEST				
TEST		Executes internal diagnostic procedure for testing movement of the axes and operation of controller I/Os.	DIRECT	

I/O Control Commands

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

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
DISABLE				
	DISABLE {IN/OUT} *n*	Disconnects the physical input or output from normal system control.	DIRECT	$1 \le n \le 16$
	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	$1 \le n \le 16$ ENABLE is default mode.
FORCE				
	FORCE {IN/OUT} *n* {0/1}	Forces a disabled input or output to a different state.	DIRECT	$1 \le n \le 16$ 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 the 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	$1 \le n \le 16$ 0=OFF; 1=ON
IF				
	IF IN[*n*]={0/1}	Checks the state of an input.	EDIT	$1 \le n \le 16$ 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	$1 \le n \le 16$ 0=OFF; 1=ON

Program Control Commands

RUN	PRIORITY	PEND
A		POST
STOP	SET *var* TIME	QPEND
SUSPEND		QPOST
CONTINUE	DELAY	
	WAIT	
	TRIGGER BY IN/OUT	

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	$1 \leq var \leq 10$. 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	$1 \le n \le 16$ 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 var3 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	HERE	SETP
DIMP	HEREC	
	HERER	TOOL
DELP		
UNDEF	TEACH	ATTACH
	TEACHR	
DELETE		SET *var*=PVAL
INSERT	SETPV	SET *var*=PVALC
	SETPVC	SET *var*=PSTATUS
	SHIFT	
	SHIFTC	

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	$1 \le axis \le 12$
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	$1 \le axis \le 12$
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	
DELETE			
DELETE &*pvect*[*n*]	Deletes coordinates for position *n* in vector &*pvect*; all recorded positions above *n* are moved down one place until an unrecorded position is encounted.	DIRECT, EDIT	Vector name must have prefix &.
INSERT			
INSERT &*pvect*[*n*]	Records current coordinates of axes, and inserts them into vector &*pvect* at position *n*; all recorded positions above *n* are moved up one place until an unrecorded position is encountered.	DIRECT, EDIT	Vector name must have prefix &.

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
HERE				
	HERE *pos*	Records joint coordinates for current position of axes.	DIRECT, EDIT	Joint coordinates = encoder counts.
HEREC				
	HEREC *pos*	Records Cartesian coordinates for current position of robot.	DIRECT, EDIT	Cartesian coordinates: XYZ in linear units; pitch/roll in angular units.
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	
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	$1 \leq axis \leq 12$
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	$1 \leq axis \leq 12$
	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}

COMMAND FORMAT	DESCRIPTION	MODE	NOTES
SETP			
SETP *pos2=pos1*	Copies the coordinates and type of *pos1* to *pos2*.	DIRECT, EDIT	
TOOL			
TOOL *length offset angle*	Defines the position of the tool center point (TCP) relative to the robot's flange.	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	
SET			
SET *var*=PVAL *pos axis*	Assigns *var* the value of one joint coordinate of a recorded position.	DIRECT, EDIT	$1 \leq axis \leq 12$
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 DIM DELVAR
GLOBAL DIMG PURGE

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 program.
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	
PURGE			
PURGE	Deletes all unused variables from user RAM.	DIRECT	
SET (mathematical and logical functions)			
SET *var1=var2*	Assigns the value of *var2* to *var1*.	DIRECT, EDIT	
SET *var1=oper var2*	Performs operation on *var2* and assigns result to *var1*.	DIRECT, EDIT	*oper*: ABS, NOT
SET *var1=var2 oper var3*	Performs operation on *var2* and *var3* and assigns result to *var1*.	DIRECT, EDIT	*oper*: +, –, *, /, AND, OR, EXP, LOG, MOD, SIN, COS, TAN, ATAN,
SET *var1*=COMPLEMENT *var2*	Inverts each binary bit of *var2* and assigns the result to *var1*.	DIRECT, EDIT	
SET *var*=PAR *n*	Assigns the value of the specified parameter to *var*.	DIRECT, EDIT	

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 *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	INIT EDITOR	PASSWORD
LET PAR	INIT CONTROL	PRIV[ILEGE]
SHOW PAR		

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 parameter values.
PASSWORD				
PASSWORD		Activates procedure for changing the password which protects the PRIVILEGE mode.	DIRECT	
PRIV[ILEGE]				
PRIV ON PRIV OFF		Activates and cancels the PRIVILEGE mode.	DIRECT	Requires password.

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.	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	$1 \leq axis \leq 12$
	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, current line number and command being executed.	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	

`SEND prog > PRN:`
`SENDPROG > PRN:`
`SENDPOINT > PRN:`
`SENDVAR > PRN:`
`SENDPAR > PRN:`
	Prints list at a printer connected to controller's parallel port.	DIRECT	

User Interface Commands

QUIET	PRINT	HELP
NOQUIET	PRINTLN	
		DO
ECHO	READ	
NOECHO	GET	ENGLISH
		JAPANESE

COMMAND	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	
PRINT				
	PRINT "*string*"	Displays *string* on screen. s).	DIRECT, EDIT	
	PRINT *var1...var4*	Displays value(s) of specified variable(s).	DIRECT, EDIT	
PRINTLN				
	PRINTLN	Same as PRINT, but starts a new line before displaying text.	EDIT	
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	

COMMAND	FORMAT	DESCRIPTION	MODE	NOTES
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	
DO				
DO *editcom*		Executes certain EDIT mode commands when controller in DIRECT mode.	DIRECT	
ENGLISH				
ENGLISH		Causes controller messages to be displayed in English on screen.	DIRECT	
JAPANESE				
JAPANESE		Causes controller messages to be displayed in Japanese on screen.	DIRECT	

Program Manipulation Commands

COPY RENAME REMOVE EMPTY 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	
EMPTY				
	`EMPTY` *`prog`*	Deletes all program lines, but leaves program existent and valid.	DIRECT	
EDIT				
	`EDIT` *`prog`*	Activates EDIT mode for program creation and editing.	DIRECT	

Editing Commands

S	*	END	
P	@		
L	EXIT		
DEL			
<Enter>			

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		Not a user command.
	(END)	End of listing. Automatically displayed by system.		Not a user command.

RS232 Communication Commands

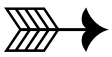
SENDCOM	PRCOM	CLRCOM
GETCOM	PRLNCOM	
	READCOM	

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

In addition, **Controller-B** has a password-protected **PRIVILEGE** mode. Most of the system parameters, and some commands, can be accessed only when the PRIVILEGE mode is active.

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.

Some 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 a teach pendant is not available, you can assume direct control of the robot and peripheral axes from the keyboard by activating Manual mode. Manual mode can be activated only when the system is operating in DIRECT mode.

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

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

Refer to the command ~ (Manual Control) at the end of 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/teach pendant.

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.
↑ and ↓	Repeats the last command(s) entered.

PRIVILEGE Mode

Most of the parameters and some **ACL** commands for **Controller-B** can be accessed only when the controller's PRIVILEGE mode is active.

The status of parameter 19 indicates whether or not the PRIVILEGE mode is active. If PAR 19=0, a password is required to access the protected parameters and commands; if PAR 19=1, no password is required.

Refer to the commands PASSWORD and PRIV[ILEGE] in Chapter 3 for more information on the PRIVILEGE mode.

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.

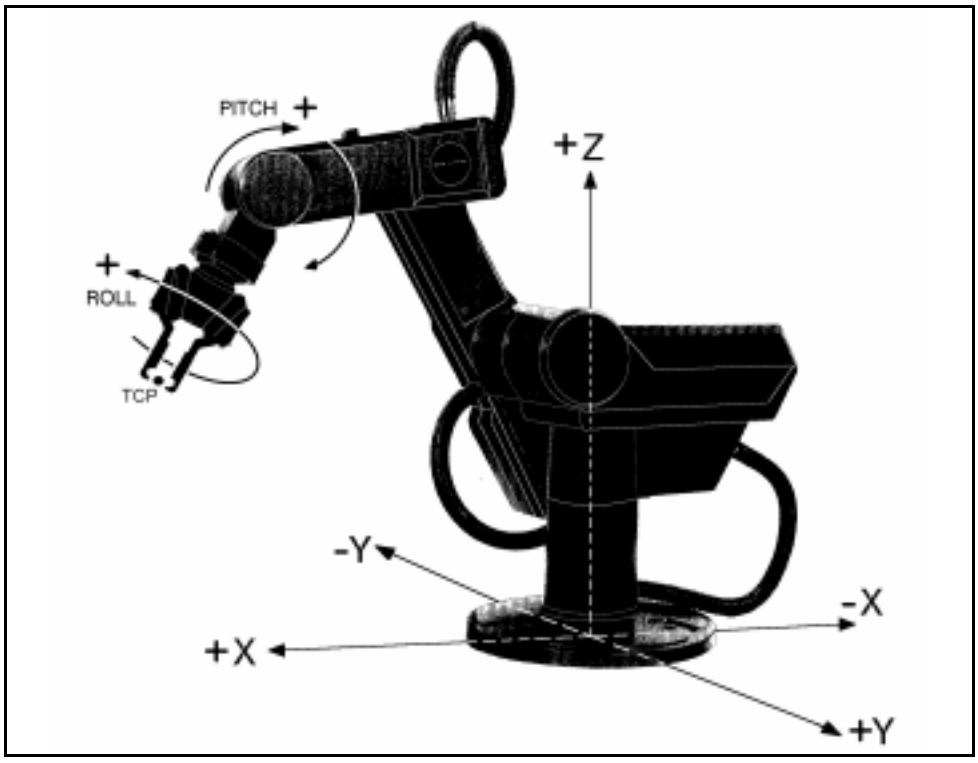
Cartesian (XYZ) Coordinates

The Cartesian, or XYZ, coordinate system is a geometric system used to specify the position of the robot's TCP (tool center point) 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 diagram below.

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

The TOOL command (or parameters 308, 309 and 310) defines the exact location of the TCP.

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



TCP Cartesian Definition

World (XYZ) Space

Certain restrictions apply when recording Cartesian position coordinates and when programming and executing XYZ movement commands (MOVEL, MOVEC and Linear SPLINE). The validity of these positions and movements is determined by three areas of world (XYZ) space. Refer to the diagrams on this page and the next.

World Space A

- **SCORBOT-ER IX**, **PERFORMER MK2**: Positions in which link 3 is at a negative angle relative to link 2 of the robot arm.
- **SCORBOT-ER 14**: Positions in which link 2 is at a negative angle relative to link 1 of the robot arm.

World Space B

- **SCORBOT-ER IX**, **PERFORMER MK2**: Positions in which link 3 is at a positive angle relative to link 2 of the robot arm.
- **SCORBOT-ER 14**: Positions in which link 2 is at a positive angle relative to link 1 of the robot arm.

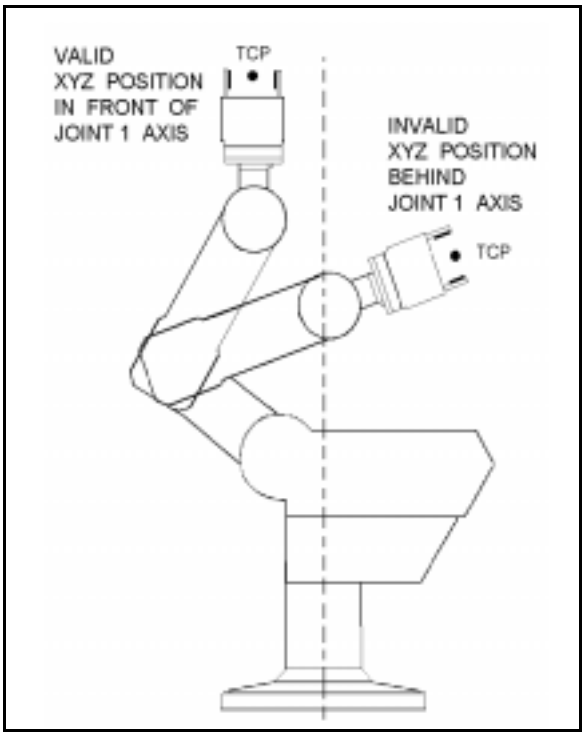
All position within World Space A and B can be recorded in Cartesian coordinates, and reached by the robot's TCP.

The XYZ movement commands (MOVEL, MOVEC and Linear SPLINE) are allowed within both World Space A and B. However, all positions referenced in the command must belong to *either* World Space A or World Space B. If the command implies a movement from one space to the other, an error message is displayed, and the command is aborted.

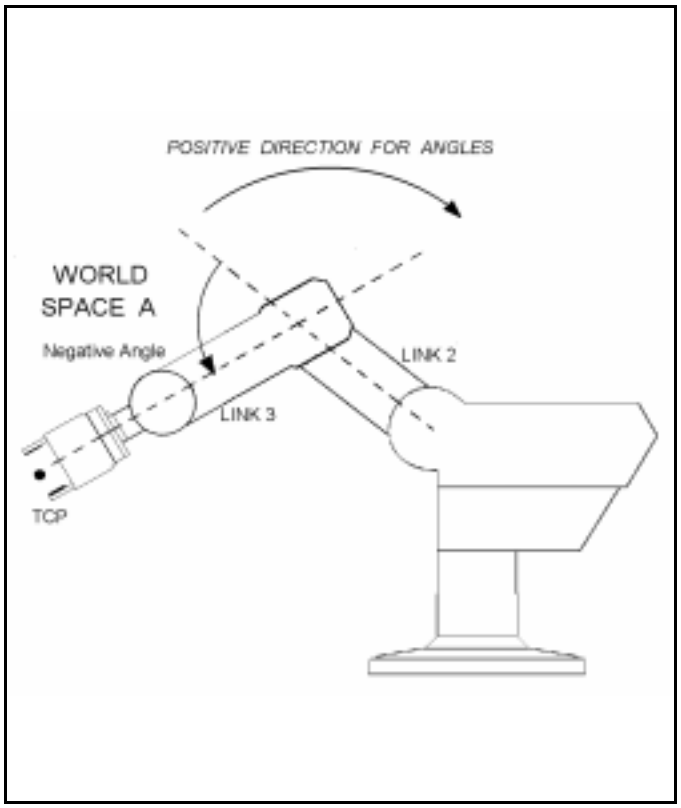
Invalid XYZ Space

For the **SCORBOT-ER IX** and **PERFORMER MK2**, positions behind the line of axis 1 cannot be recorded in Cartesian coordinates and cannot be reached by the robot's TCP by means of XYZ movement commands (MOVEL, MOVEC and Linear SPLINE). These positions can, however, be recorded and reached in the Joint mode.

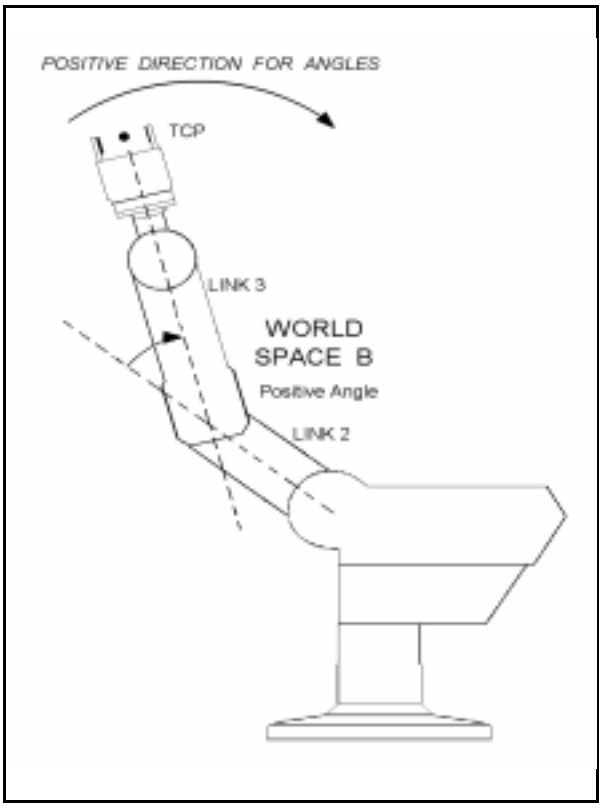
This invalid XYZ posture is termed World Space C.



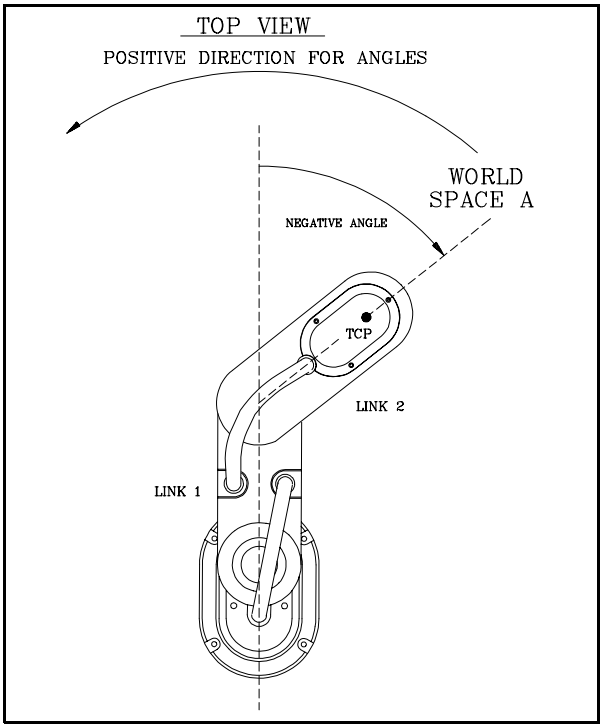
World Space C



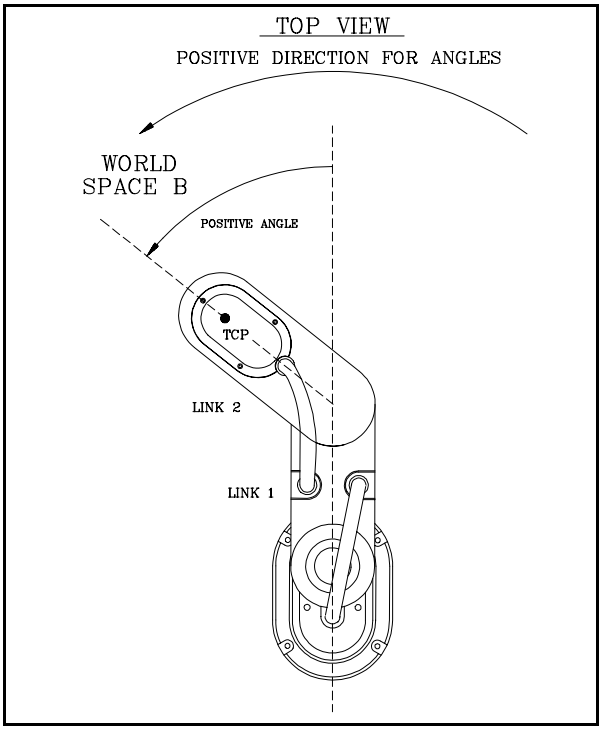
ER IX and MK2: World Space A



ER IX and MK2: World Space B



ER 14: World Space A



ER 14: World Space B

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.

Other than the limitations of the robot's working envelope, no restriction apply when recording joint position coordinates and when programming and executing joint movement commands (MOVE, MOVES and Joint SPLINE).

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 defined in either DIRECT or EDIT mode, and 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 variables can be defined only in EDIT mode, and can only be used in the program which was being edited at the time the private variable was defined.
  - The command DEFINE is used to define a private variable.
  - The command DIM is used to define an array of private variables.

Up to 12 variables can be defined in one command.

Names used to define variables may be a combination of up to five alphanumeric 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 X=`**`CPOS`**`[`*n*`]-`**`ENC`**`[`*n*`]`	(where *n* is an axis number) The valueof the position error of the specified axis is the difference between system variables CPOS and ENC; this value is assigned to the variable X.
`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.

ACL for **Controller-B** contains 14 system variables:

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

The indices indicate the dimensions of the array variables.

- IN, ENC, HS, ZP, CPOS, TIME, LTA, LTB, and MFLAG are read-only variables.
- ERROR, ERRPR, ERRLI, OUT are read-write variables.
- ANOUT is a read-write variable only in PRIVILEGE mode; otherwise it is read-only.

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)
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 –2147483647 to +147483647 to define the coordinates, and one word in the range –32668 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. Refer also to the chart of position recording commands later in this chapter.

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

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

- **Absolute XYZ** (HEREC, 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 of one position and encoder values of 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 of one position and the Cartesian coordinate values of 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 of one position to the encoder values of 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 of one position to the Cartesian coordinate values of 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.

  Robot (group A) positions with this type of name do not need to be defined before they are recorded; the position recording commands automatically define and record numerically named robot positions.

- 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.

  Positions 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.

  Position vectors whose names are prefaced by the character & can be manipulated by means of DELETE and INSERT commands.

  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, HEREC, HERER, TEACH, TEACHR, SETPV, SETPVC, SHIFT and SHIFTC are used to record position coordinate values.

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	`HEREC pos`	DIRECT EDIT
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, according to user settings.

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. Most parameters are password-protected.

Parameters are referred by their number (1 to 699). 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*	Position: *pos*
Variables: *var*	Position vector: *pvect*
Value: *value*	Duration (time): *duration*
Axis: *axis*	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 the display of data resulting from SHOW ENCO, LIST, SEND, and other commands.

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, and PRIVILEGE.
- 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* and *var2* are variables or 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.
```

Notes: Refer to the IF command.

APPEND DIRECT

Format: APPEND

Description: APPEND loads data from a backup file in the host computer to the controller's user RAM, via the main RS232 channel (**Controller-B**'s CONSOLE port).

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.

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*

Also refer to the SEND and RECEIVE commands.

DIRECT ATTACH

Format:

ATTACH *pvect*

ATTACH OFF{A/B/C}

ATTACH ?

Where: *pvect* is a vector.

Description:

ATTACH *pvect*	Attaches the specified position vector to the teach pendant according to the group for which the position vector is defined. When a vector is attached to the teach pendant, all references to that group refer to the positions in the attached vector. Only one vector at a time may be attached to each group. Attaching another position vector cancels the previous attachment for this group.
ATTACH OFFA ATTACH OFFB ATTACH OFFC	Detaches the position vector from teach pendant according to the group specified.
ATTACH ?	Displays the current ATTACH status.

Examples:

- DIMP ALPHA[20]
ATTACH ALPHA

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

- DIMPB &BETA[30]
ATTACH &BETA

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

- ATTACH OFFB

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

AUTO DIRECT

Format: AUTO

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

AUTO transfers control from the teach pendant to the keyboard.

DIRECT/EDIT CLOSE

Format: `CLOSE [`*`var`*`]`

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

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

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

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

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

Examples:

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

Notes: Refer to the OPEN and JAW commands.

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

Also refer to the gripper parameters in Chapter 7.

CLR — DIRECT:PRIVILEGE

Format: `CLR` *n*

Where: *n* is an encoder, $1 \le n \le 12$, or *.

Description:

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

Encoder values can be cleared only when the PRIVILEGE mode is active.

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.

Notes: Refer to the section on PRIVILEGE mode in Chapter 2.

Refer to the PRIV command.

CLRBUF

Format:

```
CLRBUF[A/B]
CLRBUF n
```

Where: *n* 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 n`	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 EDIT

Format: `CLRCOM [`*n*`]`

Where: *n* is an RS232 communication port, $0 \le n \le 8$

Description:

`CLRCOM`	Clears the buffers of all the RS232 communication ports.
`CLRCOM` *n*	Clears the buffers of the specified RS232 port.

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.

Examples:

- `CLRCOM 2` Clears the buffers of RS232 port COM2.
- `CLRCOM 0` Clears the buffer of **Controller-B**'s RS232 port COM0.

Note: Refer to the SENDCOM command.

DIRECT COFF

Format: `COFF[A/B]`

`COFF` *n*

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

Description:

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

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

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

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

COFF must be activated before you change parameters values.

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

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

```
CONTROL DISABLED
```

Examples:

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

Note: Refer to the CON command.

CON DIRECT

Format: `CON[A/B]`

`CON` *n*

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

Description:

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

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

```
CONTROL ENABLED
```

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

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

Examples:

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

Note: Refer to the COFF command.

DIRECT CONFIG

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! Also erases current password and resets to factory-set default.

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.

Example:

```
>config
!!!WARNING ALL USER PROGRAMS WILL BE ERASED.
ARE YOU SURE ??? [YES/NO] > yes <Enter>
JOB KILLING PHASE ..........>
ENTER NUMBER OF INPUTS [16] (0-16) ><Enter>
ENTER NUMBER OF OUTPUTS [16] (0-16) ><Enter>
ENTER NUMBER OF ENCODERS [8] (0-12) ><Enter>
ENTER NUMBER OF DACS [8] (0-12) ><Enter>
ENTER NUMBER OF AUXILIARY RS232 PORTS[0] (0-8) ><Enter>
WHICH TYPE OF ROBOT (0-NONE,2-MK2,9-ERIX,14-SCORA) [9]
                                        (0-14) ><Enter>
ENTER NUMBER OF SERVO LOOPS, GROUP A [5] (5-8) ><Enter>
SERVO GRIPPER INSTALLED AT AXIS [6] (0-8) ><Enter>
ENTER NUMBER OF SERVO LOOPS, GROUP B [2] (0-2) ><Enter>
ENTER TOTAL NUMBER OF SERVO LOOPS [8] (8-8) ><Enter>
ENTER AMOUNT OF BATTERY BACKED RAM IN KBYTES [512]
                                        (132-512) ><Enter>
ENTER NUMBER OF USER PROGRAMS [80] >400 <Enter>
ENTER NUMBER OF USER PROGRAM LINES [1200] >4000 <Enter>
ENTER NUMBER OF USER VARIABLES [300] >4000 <Enter>
ENTER NUMBER OF USER POINTS , GROUP A [900] >4000 <Enter>
ENTER NUMBER OF USER POINTS , GROUP B [850] >4000 <Enter>
ENTER NUMBER OF USER POINTS , GROUP C [0] > <Enter>
ENTER NUMBER OF USER COMMENTS [200] >1000 <Enter>
Performing configuration, please wait 10 seconds

Available workspace 393788 (Bytes) 100%
```

```
Assigned workspace  298021 (Bytes)  74%
Unused workspace     95767 (Bytes)  24%
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 12, 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.
  - 2: Compatible with **PERFORMER-MK2** kinematics.
  - 9: Compatible with **SCORBOT-ER IX** kinematics.
  - 14: Compatible with **SCORA-ER 14** kinematics.
- The number of axes for groups A and B are user definable. When the robot is defined as Type 9 or 2, group A must be the robot and include a minimum of 5 axes. When the robot is defined as Type 14, group A must be the robot and include a minimum of 4 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 group A includes 4 axes, the gripper must be installed as axis 5. If no servo gripper is installed, enter 0 as the gripper axis. You can then use the (gripper) axis 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, totalling 8 axes.
- The standard size of user RAM is 512Kb.
- The number of user defined programs, program lines, variables, positions, and comments depends upon the memory size and the allocation of all these items.

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

The workspace allocations are as follows:

- Available Workspace: Memory remaining available to user after memory allotted to controller.
- Assigned Workspace: Percentage of the Available Workspace. Calculated according to the setttings you entered during the configuration.

- Unused Workspace: Percentage of the Available Workspace not allotted.

Note that there is a delay of several second before the controller displays OK.

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 shown above.

```
>CONFIG ?

   ********  CURRENT CONFIGURATION IS :
ER-IX ROBOT TYPE
GRIPPER ON AXIS  6
16 INPUTS               | 512   KB BACKED-UP MEMORY
16 OUTPUTS              | 400   PROGRAMS
8  ENCODERS | 4000  PROGRAM LINES
8  ANALOG OUTPUTS       | 4000  VARIABLES
0  AUXILIARY PORTS      | 4000  POINTS IN GROUP A
5  AXES IN GROUP A      | 4000  POINTS IN GROUP B
2  AXES IN GROUP B      | 0     POINTS IN GROUP C
8  TOTAL AXES           | 1000  COMMENTS

Available workspace 393788 (Bytes) 100%
Assigned workspace  298021 (Bytes)  74%
Unused workspace     95767 (Bytes)  24%
```

CONTINUE DIRECT/EDIT

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.

DIRECT COPY

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 is displayed.

Example:

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

DEFINE EDIT

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

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

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

Up to twelve variables can be defined in one command.

Examples:

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

Note: This command does not create a program line.

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.

The DEFP command is not required for *numerically* named positions for group A; these positions will be automatically defined when entered as part of a command.

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.

DEL EDIT

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.

EDIT DELAY

Format: `DELAY` *var*

Where: *var* is a variable or constant.

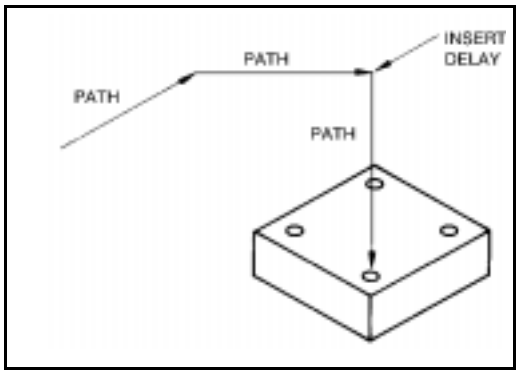
Description: Delays the execution of a program.

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

The DELAY command is used for the following purposes:

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

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



Examples:

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

DELETE DIRECT

Format: `DELETE &pvect[n]`

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

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

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

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

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

Example:

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

Note: Refer to the DIMP and INSERT commands.

DELP

Format:

`DELP` *`pos`*

`DELP` *`pvect`*

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

Description:

Deletes positions and position vectors from user RAM.

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

The DELP command cannot delete individual positions within a vector.

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

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

Examples:

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

Notes:

This command does not create a program line.

Refer to the DELETE and UNDEF commands.

DELVAR DIRECT/EDIT

Format: DELVAR *var*

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

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

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

In DIRECT mode DELVAR deletes only global variables.

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

You cannot delete system variables.

Examples:

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

Note: This command does not create a program line.

Format: `DIM var[n]`

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

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

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

Example:

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

Note: This command does not create a program line.

DIMG DIRECT/EDIT

Format: `DIMG var[n]`

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

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

A global variable can be used by any user program.

Example:

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

Note: This command does not create a program line.

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

`DIMPC pvect[n] axis`

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

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

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

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

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

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

Examples:

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

Notes: This command does not create a program line.

DIR DIRECT

Format: `DIR`

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

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

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

Example: ■

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

Notes: Validity: Refer to the EXIT command.

Priority: Refer to the PRIORITY and RUN commands.

DISABLE

Format:

```
DISABLE {IN/OUT} n

DISABLE ?
```

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

Description:

`DISABLE IN n` `DISABLE OUT n`	Disconnects the physical input or output from normal system control.
`DISABLE ?`	Displays a list of 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
QPOST	PRCOM
PRINTLN	PRLNCOM
READ	READCOM
STOP	SENDCOM

Example:

- `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.

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.

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.

EMPTY DIRECT

Format: `EMPTY` *prog*

Where: *prog* is a user program.

Description: Deletes all lines of a program, but leaves the program existent and valid.

The EMPTY command is useful when *prog* is an application subroutine which is called by another program.

Once a program has been emptied, you can alter its specific function without changing the main program. This is particularly useful in CIM applications, when you want to change a process at a given station without changing the CIM system programs.

Private variables assigned to this program are also deleted.

Example: ■

```
    PROGRAM MAIN
    ****************
[program line]
[program line]
[program line]
    .
GOSUB USER
    .
[program line]
[program line]
[program line]
    .
END

    PROGRAM USER
    ***************
[program lines which]
[can be removed and]
[rewritten by means of]
[the EMPTY command]

END
```

Use the command EMPTY USER to erase all program lines. Then EDIT program USER to alter the contents of the program.

Format: `ENABLE {IN/OUT} n`

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

Description:

`ENABLE IN n` `ENABLE OUT n`	Restores normal system control of an input or output which has been disconnected by means of the DISABLE command.

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 EDIT

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

The system automatically writes (END) at the end of a listing.

END is not a user command.

Format: `ENDFOR`

Description: Required companion to FOR command.

Ends the subroutine to be executed by the FOR command.

Example:

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

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

Note: Refer to the FOR command.

ENDIF EDIT

Format: `ENDIF`

Description: Required companion to IF command.

Ends the subroutine to be executed by the IF command.

Example:

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

Note: Refer to the IF command.

DIRECT ENGLISH

Format: `ENGLISH`

Description: Causes the controller messages to be displayed in English.

If garbled text appears on the screen, use this command to make the system communicate in English.

Note: Refer to the JAPANESE command.

<Enter> EDIT/DIRECT

Format: `<Enter>`

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

In DIRECT mode, confirms and executes the command.

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

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

EXACT

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

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

`MOVED`	`MOVELD`	`SPLINED`
`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 on for group A.
- `EXACT OFFA` — EXACT off for group A.

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

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

EXIT EDIT

Format: `EXIT`

Description: Quits EDIT mode 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.

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, $1 \le n \le 16$;
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
```
Forces input 5 to ON state.

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

-
```
>force ?
INput[5]=1
OUTput[11]=0
```
Displays status of forced I/O's:
Input 5 is in forced ON state.
Output 11 is in forced OFF state.

Note: Refer to the DISABLE command.

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

GET EDIT

Format: GET *var*

Where: *var* is a user variable.

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

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

Example:

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

Note: Refer to the READ command.

Format: `GETCOM n var`

Where: *n* is an RS232 communication port, $0 \leq n \leq 8$;
var is a variable.

Description: Companion to the SENDCOM 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 SENDCOM command.

GLOBAL DIRECT/EDIT

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

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

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

Examples:

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

Note: This command does not create a program line.

Format: GOSUB *prog*

Where: *prog* is a user program.

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

Example:

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

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

GOTO EDIT

Format: GOTO *labeln*

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

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

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

Examples:

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

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

- This program is executed 500 times and then stops.

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

Note: Refer to the LABEL command.

HELP

Format: `HELP [topic]`

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

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

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

This command does not create a program line.

HERE / HEREC DIRECT/EDIT

Format:

`HERE` *pos*

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

`HEREC` *pos*

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

Description:

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

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

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

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

Examples:

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

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

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

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

Format:

`HERER` *`pos2`* DIRECT mode only.

`HERER` *`pos2 pos1`*

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

Description: HERER allows you to record a position relative to another position, or relative to the current position.

`HERER` *`pos2`*

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

You must 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 values of *pos2*, relative to *pos1*, in joint values.

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.

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

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

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 values displayed in brackets are the offset values last recorded for this position. If no values have been recorded, the bracket is empty [.] .

- `>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]`

  Defines 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, $1 \le n \le 12$.

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 2, 9 or 14 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 all axes in a particular configuration.
`HHOME n`	Drives the specified axis to its home position by searching for a hard stop. HHOME is used for a device, such as a linear slidebase, which does not have a microswitch. (Note: When hard homing a slidebase, the moving base must be near enough to the mechanical end stop of the LSB for the homing to succeed.)

The robot axes *must be homed at the beginning of each working session*. Otherwise, movement and position recording commands will not be executed.

Homing of the axes is not required before movement and position recording commands only if both PAR 460+*axis*=0 and PAR 600+*axis*=0 .

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 robot has not been homed, the **Controller-B** ignores the axes' software (encoder) limits; you can move the individual axes, by means of the teach pendant or keyboard, to any location within their range.

The robot must be homed before an end effector (gripper or tool) is mounted on the end-of-arm flange.

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. The following message is displayed:

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

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

```
HOMING COMPLETE (ROBOT)
```

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

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

The coordinate of the roll axis is constantly copied into an internal battery backed up variable LAST_ROLL. The LAST_ROLL value is used by the HOME procedure to return the roll to the approximate home position prior to homing immediately after powering on the system. This prevents the gripper cable from becoming entangled or torn during the homing procedure.

The updating of LAST_ROLL is cancelled (and LAST_ROLL is set to 0) when any of the following occurs:

- You move the roll axis from the teach pendant or keyboard after powering on the system and before homing.
- You abort the HOME procedure before its completion.
- Controller configuration.

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 axis 7.

Notes:

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

[RUN] 0 [ENTER]

Refer to the homing parameters in Chapter 7.

Also refer to your robot's *User's Manual* for more information on the Homing routine.

Format: `IF var1 oper var2`

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

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

Examples:

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

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

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

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

INIT DIRECT

Format:
```
INIT CONTROL
INIT EDITOR
```

INIT CONTROL

Description: Initializes all system control parameters.

INIT CONTROL must be executed after any changes are made to the values of 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 all user RAM, except parameters.

The CONFIG command automatically performs this operation.

INSERT

Format: `INSERT &pvect[n]`

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

Description: Records the current coordinates of the robot or device (same as HERE or HEREC commands) for position &*pvect*[*n*] and inserts the coordinates into the vector at index *n*. The inserted position is recorded according the type (JOINT or XYZ) of position whose coordinates currently occupy &*pvect*[*n*].

All recorded positions above position &*pvect*[*n*] are then moved up one place until an empty (unrecorded) position is encountered. If all positions in the vector above &*pvect*[*n*] are recorded, an error message is displayed and the command is cancelled.

This command provides the means for modifying an existing SPLINE or MOVES trajectory, and for inserting a new position within a vector.

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

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

Examples:

- `INSERT &AA[4]` — Records position 4 in vector &AA; Inserts this position in vector &AA at position &AA[4]; the previous position &AA[4] now becomes &AA[5].
- `HERE &AA[6]` — If coordinates have not yet been recorded for position &AA[6], insertion is not required.

Note: Refer to the DELETE, SPLINE, and MOVES commands.

INT_ON / INT_OFF DIRECT/EDIT

Format: `INT_ON` *n1* [*n2 ... n4*]

`INT_OFF` *n1* [*n2 ... n4*]

Where: *n1 . . . n4* 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 cancelling 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.

DIRECT JAPANESE

Format: JAPANESE

Description: Causes the controller messages to be displayed in Japanese.

If system messages are displayed on the screen in English, use this command to make the system communicate in Japanese.

Note: Refer to the ENGLISH command.

JAW DIRECT/EDIT

Format: `JAW var [duration]`

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

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

Duration is defined in hundreths of a second.

The JAW command can be used only for servo grippers.

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

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

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

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

Examples:

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

Note: Refer to the CLOSE and OPEN commands.

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, $0 \leq labeln \leq 9999$.

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;
var is a variable or constant.

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

Most parameters can be changed only when the PRIVILEGE mode is active. The following parameters can be accessed during normal operation:

- Gripper parameters: 73, 74, 75, 76, 274 and 275.
- Smoothing parameter: 219
- Trajectory parameter: 220 and 236
- Position Error parameters: 261-272

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

```
INIT CONTROL
```

Examples:

- `LET PAR 73=9355` Sets parameter 73 to 9350.
  `INIT CONTROL`

- `LET PAR 261 100` Sets parameter 261 to 100.
  `INIT CONTROL`

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

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

Refer to the PRIV command.

LIST DIRECT

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

Where: *prog* is a program.

Description:

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

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

Example: ■ `>`**`LIST AAA`** Displays all lines in program AAA.

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

Note: Refer to the SEND commands.

Format: LISTP

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

Positions with numeric names are not listed.

Example:

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

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

and 13 additional numerical points defined.
```

LISTPV DIRECT

Format: `LISTPV` *pos*

`LISTPV POSITION`

Description:

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

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

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

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

Example:

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

DIRECT LISTVAR

Format: LISTVAR

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

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

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

Example: ■ >**LISTVAR**

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

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

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

MODULO ROLL — DIRECT/EDIT

Format: `MODULO ROLL`

Description: Returns the value of the roll axis to a value within the range of ±360°, without moving the roll axis.

MODULO ROLL enables unlimited rotations of the roll axis, by preventing the (software/encoder) axis limits from being reached. It is therefore a useful when an application requires an end effector, such as a screwdriver, to move continuously in one direction.

MODULO ROLL is not executed until the movement buffer of the roll axis is empty, so as not to affect previously issued MOVE commands. Thus, a program which issues a MODULO ROLL command will be suspended until the roll axis movement buffer is empty.

This command enables continuous rotation of an end effector, such as a screwdriver.

Warning! Use this command with caution when cables or hoses are connected to the end effector (such as a gripper). Be sure the cables or hoses will not become improperly stretched or entangled following a MODULO ROLL command.

Example:

```
HERE POS1                 Assuming roll value for pos1 is 0°:
LABEL 1
SHIFTC POS1 BY R 190
MOVE POS1                 Roll rotates +190°;
SHIFTC POS1 BY R 190
MOVE POS1                 Roll again rotates +190°, reaching 380°;
MODULO ROLL               MODULO ROLL returns 380° to 20°;
SHIFTC POS1 BY R 190      Roll again rotates +190°, reaching +210°.
MOVE POS1
GOTO  1
```

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

Format:

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

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

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

MOVE

Description:

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

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

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

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

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

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

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

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

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

- Use the MOVED command.

MOVED

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

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

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

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

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

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

MOVE, MOVED Summary

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

Examples:

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

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

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

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

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

Note: Refer to the EXACT command.

MOVEC / MOVECD DIRECT/EDIT

Format: `MOVEC` *`pos1 pos2`*

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

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

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

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

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

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

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

Examples:

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

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

Note: Refer to the SPEEDL command.

DIRECT/EDIT MOVEL / MOVELD

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

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

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

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

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

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

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

Example:

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

Note: Refer to the SPEEDL command.

MOVES / MOVESD DIRECT/EDIT

Format:

`MOVES` *`pvect n1 n2`* [*`duration`*]

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

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

Description:

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

All positions in the vector must be absolute joint positions.

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

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

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

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

Example:

■

```
MOVED PATH[1]
MOVESD PATH 2 20
MOVESD PATH 19 1
```

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

Note:

Refer to the SPLINE and SPLINED commands.

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 DIRECT

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.

DIRECT/EDIT OPEN

Format: `OPEN [var]`

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

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

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

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

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

Examples:

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

Notes: Refer to the CLOSE and JAW commands.

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

Also refer to the gripper parameters in Chapter 7.

ORIF EDIT

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

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

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

Example:

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

Note: Refer to the IF command.

EDIT P

Format: P

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

PASSWORD DIRECT

Format: PASSWORD

Description: A password is required in order to activate the PRIVILEGE mode.

This command allows you to change the password which protects the PRIVILEGE mode.

```
>PASSWORD
ENTER PRESENT PASSWORD:
ENTER NEW PASSWORD:
ENTER AGAIN:
```

You are prompted to do the following, in sequence:

- Enter the currently defined password.
- Enter the new password; it may contain up to 8 characters.
- Again enter the new password.

The controller is factory-set to accept <Enter> as the password.

Following a controller configuration, the currently defined password is erased and reset to the factory-set default.

Note: Refer to the PRIV command.

Format:

PEND *var1* FROM *var2*

POST *var3* TO *var2*

Where: *var1* is a variable;
var2 is a global variable;
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

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

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

PRCOM EDIT

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

Where: *n* is an RS232 communication port, 0≤*n*≤8;
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: Displays strings and variable values on screen.

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

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

A variable is one argument, regardless of length.

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

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

Example:

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

Will display:

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

Note: Refer to the PRINT command.

PRIORITY

Format: `PRIORITY` *`prog var`*

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

Description: Sets the priority of program *prog* to the value of *var* .

Priority ranges 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 powered 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.

PRIV[ILEGE] DIRECT

Format: `PRIV {ON/OFF}`

`PRIVILEGE {ON/OFF}`

Description: The PRIVILEGE mode prevents access to most of the controller's parameters and several **ACL** commands. This feature prevents accidental or improper manipulation of servo and other critical parameters.

The following commands are *protected*:

```
SET ANOUT
CLR
LET PAR
```

The following parameters are *not protected*:

`PAR 73,74,75,76, 274, 275`	Gripper parameters
`PAR 219`	Smoothing parameter
`PAR 220,236`	Trajectory parameters
`PAR 260+`*`axis`*	Position error parameters

To activate the PRIVILEGE mode, use the command:

```
PRIV ON
```

You are then prompted to enter the password.

Once the PRIVILEGE mode is active, you may manipulate the protected parameters and commands.

To cancel the PRIVILEGE mode, use the command:

```
PRIV OFF
```

Notes: Refer to the PASSWORD command, and to the section, "Privilege Mode," in Chapter 2.

Warning! Only experienced users should attempt parameter manipulation. Refer to Chapter 7 for information on system parameters, and heed all warnings given there.

EDIT PRLNCOM

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

Where: *n* is an RS232 communication port, 0≤*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.

PROFILE DIRECT/EDIT

Format: `PROFILE PARABOLE {A/B/C}`

`PROFILE SINUS {A/B/C}`

Description: For better path performance, **Controller-B** offers two trajectory control profiles: sinusoid and paraboloid. The paraboloid profile causes the motors to accelerate slowly until maximum speed is reached, then decelerate at the same rate. The sinusoid profile causes the motors to accelerate and decelerate quickly at the start and end of movement, with a constant speed along the path.

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

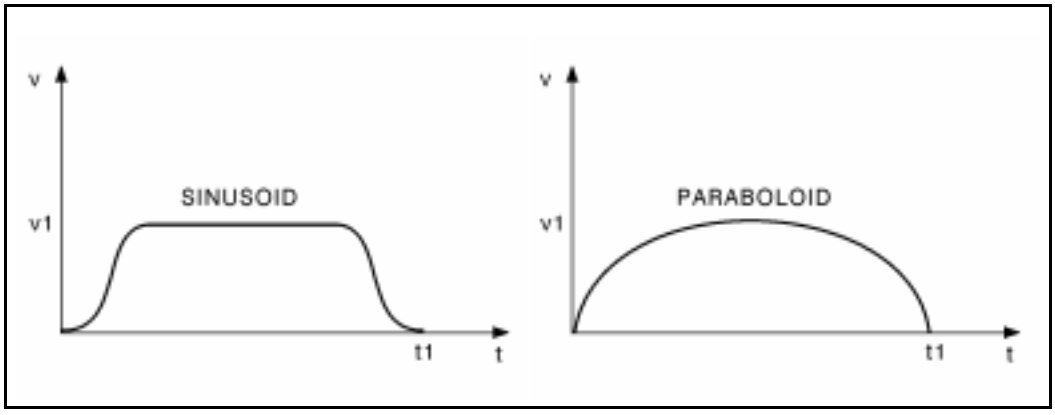
Paraboloid profile is most suitable for applications which do not require constant speed, since it does not overstress the motors.

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

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

Example:

- `PROFILE PARABOLE A` Changes robot movement profile to PARABOLE.



PSTATUS

Format: SET *var*=PSTATUS *pos*

Where: *var* is a variable;
pos is a position.

Description: Assigns *var* a value according to the type of the specified position, based on the following definitions:

Value = Type of Position
0 = Position defined, but coordinates not recorded
1 = Absolute Joint
2 = Absolute XYZ – World A
3 = Absolute XYZ – World B
4 = Relative by Joint to Another Position
5 = Relative by XYZ to Another Position
14 = Relative by Joint to Current Position
15 = Relative by XYZ to Current Position

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

Example:

- `SET PV=PSTATUS A` — If A is a position whose coordinates have been recorded as relative to the current position by joint values, then PV will be assigned a value of 14.

PURGE DIRECT

Format: PURGE

Description: Deletes from user RAM all global and private variables which are not used by any program.

PVAL / PVALC

Format:

SET *var*=PVAL *pos axis*

SET *var*=PVALC *pos coord*

Where: *pos* is a robot (group A) position;
axis is an axis number;
var is a variable;
coord can be: X, Y, Z, P, R .

Description:

SET *var*=PVAL *pos axis* Assigns *var* one of the Cartesian coordinates of the specified position.

SET var=PVALC *pos coord*
Assigns *var* the joint value of the specified axis in the specified position.

The value of the Cartesian coordinate which is assigned to the variable is defined in microns. Pitch and roll values are defined in millidegrees.

Examples:

- `SET JV=PVAL BUF1 1` JV receives the joint coordinate value of axis 1 at position BUF1.

- `SET C=PVALC POSITION Y`
C receives the value of the robot's current Y-coordinate.

QPEND / QPOST EDIT

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.

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 EDIT

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

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

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

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

The READ procedure is performed sequentially for all the arguments.

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

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

```
ENTER value >>
```

Example:

- `READ "enter value of x" X`

  Will display on screen:

  `enter value of x ?`

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

Note: Refer to the PRINT command.

Format: `READCOM` *`n var`*

Where: *n* is an RS232 communication port, $0 \leq n \leq 8$;
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 DIRECT

Format: RECEIVE [*prog*]

Description: Loads data from a backup file in the host computer to the controller's user RAM, via the main RS232 channel (**Controller-B**'s CONSOLE port.)

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

RECEIVE

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

Accepts the contents of a backup file which was generated by a SEND command.

After you enter the RECEIVE command, a warning will appear, and you will be prompted to confirm the operation. If your response is YES (complete word), 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*.

Also refer to the SEND command.

REMOVE DIRECT

Format: REMOVE *prog*

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

The system will prompt for verification:

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

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

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

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

Private variables assigned to this program are also deleted.

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

Example:

- `REMOVE PALET` Deletes program PALET.

Note: Refer to the EMPTY command.

RENAME

Format: RENAME *prog1 prog2*

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

If the name *prog2* is already in use, 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 DIRECT/EDIT

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

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

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

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

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

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

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

Examples:

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

- `RUN DEMO` — Program DEMO runs at default priority 5.

- `RUN IOS 9` — Program IOS runs with a priority value of 9.

Note: Refer to the PRIORITY command.

EDIT S

Format: S [*line_n*]

Where: *line_n* is a program line number

Description:

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

SEND DIRECT

Format:

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

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.

If a printer is connected to the **Controller-B**'s parallel port, a hard copy of the data can be produced. Use the following commands:

`SEND > PRN:`	Prints all user data (programs, positions, parameters and variables)
`SENDPAR > PRN:`	Prints a list of all parameters.
`SENDPOINT > PRN:`	Prints a list of all positions.
`SENDPROG > PRN:`	Prints all programs (includes positions and variables).
`SENDVAR > PRN:`	Prints a list of variables.

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*.

Also refer to the RECEIVE and APPEND commands.

SENDCOM EDIT

Format: SENDCOM *n var*

Where: *n* is an RS232 communication port, $0 \le n \le 8$;
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
   SENDCOM 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.

Format:

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

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

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

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

Description:

1. `SET var1=var2` — Assigns the value of *var2* to *var1*.
2. `SET var1=oper var2` — The operation is performed on *var2* and the result is assigned to *var1* .

If *oper* is ABS	Assigns the absolute value of *var2* to *var1*
If *oper* is NOT	Assigns the logical negative value of *var2* to *var1*. If *var2* ≤ 0, *var1* = 1; If *var2* >0, *var1* =0.

3. SET *var1=var2 oper var3*

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

4. SET *var1*=COMPLEMENT *var2*

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

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

6. SET *var*=PVALC *pos coord*

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

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

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

Examples:

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

SETP DIRECT/EDIT

Format: `SETP` *`pos2=pos1`*

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

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

If *pos2* has an alphanumeric name, it must first be defined using the DEFP or DIMP command. The DEFP command is not required if the *pos2* is a *numerically* named position for group A; it will automatically be defined when entered as part of the command.

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.
- Copies positions 1 through 100 from vector A to a new vector named &A. These new positions can be manipulated by DELETE and INSERT commands.

```
DEFINE I
FOR I=1-100
   SETP &A[I]=A[I]
ENDFOR
```

Format: SETPV *pos* DIRECT mode only.

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

SETPV *pos axis var*

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

SETPV *pos*

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

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

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

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

```
>SETPV P
   1 --[2388] >
   2 --[22857] >
   3 --[68120] >
   4 --[21510] >
   5 --[20825] >
```

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 coordinates are not accepted, and an error message is displayed.

Examples:

- `>HERE PQ`
  `>SETPV PQ`
  Records position PQ in joint coordinates according to the robot's current location; then permits 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 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.

Example:

- `SETPV PS 3 1000` — Changes the joint value of axis 3 for position PS to 1000.
- `SET VARP=100`
`SETPV PS 3 VARP` — Changes the joint value of axis 3 for position PS to 1000.

Note: SETPVC *pos coord var* 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 variable expressed in microns (X,Y,Z) or millidegrees (P,R);
or *var* is a constant expressed in millimeters (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.

The value of the Cartesian coordinate which is modified by this command is defined in microns (when a variable) or millimeters (when a constant). Pitch and roll values are defined in millidegrees (when a variable) or degrees (when a constant).

SETPVC will warn you of an invalid point coordinate as soon as the controller attempts to record the new coordinate.

Examples:

-
```
SET VARA=7000
SETPVC POSA Y VARA
```
The Y coordinate for robot position POSA is changed to 7 millimeters.

-
```
SETPVC POSA Y 7.000
```
The Y coordinate for robot position POSA is changed to 7 millimeters.

-
```
SETP    PA=POSITION
SETPVC  PA X 25
SETPVC  PA P -45
```
Position PA receives the coordinates values of the robot's current position. Then the value of position PA is changed by 25mm along the X axis and -45° on the pitch axis.

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 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 variable expressed in microns (X,Y,Z) or millidegrees (P,R);
or *var* is a constant expressed in millimeters (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 coordinate.

The value of the Cartesian coordinate which is modified by this command is defined in microns (when a variable) or millimeters (when a constant). Pitch and roll values are defined in millidegrees (when a variable) or degrees (when a constant).

Examples:

- `SHIFT P200 BY 1 3000` — Robot position P200 is offset by 3000 encoder counts along axis 1.

- `SHIFTC POS99 BY R 20.000` — Robot position POS99 is offset by 20° along the roll axis.

- `SET VV=20000`
  `SHIFTC POS99 BY R VV` — Robot position POS99 is offset by 20° along the roll axis.

Format:

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

SHOW DIN

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

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.

A screen zone is reserved for the display of encoder information.
The displayed value of all encoders is updated every 0.5 seconds, 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*

Where: *n* is an axis number, $1 \leq n \leq 12$

Description: Displays the DAC value for the specific axis 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=100
   O.K.
```

SHOW SPEED

Description: Displays all current speed settings.

- Linear Speed: Affects MOVEL(D) and MOVEC(D) and Linear SPLINE(D) commands.
- Joint Speed: Affects MOVE(D), MOVES(D), and Joint SPLINE(D) commands.
- Program: Speed of movement when command is executed from a running program.
- Manual: Speed of movement when command is executed in DIRECT mode.

Example: ■ **>SHOW SPEED**

```
Linear Speed (mm/sec)    Program    Manual
- - - - - - - - - - - - - - - - - - - - - - -
                           500       500.000

Joint Speed (%)          Program    Manual
- - - - - - - - - - - - - - - - - - - - - - -
Group A                    50         50
Group B                    50         80
Axis 9                    100         50
```

The following chart describes how these speed settings are determined.

Linear Speed: Program	**Linear Speed: Manual**
Defined by command **SPEEDL** *var* in EDIT mode. Displayed in millimeters/second.	Defined by command **SPEEDL** *var* in DIRECT mode. Displayed in millimeters/second.
	Defined by teach pendant command **SPEEDL** in XYZ mode. (Entered as a percentage of maximum linear speed.) Displayed in millimeters/second.
Joint Speed: Program	**Joint Speed: Manual**
Defined by command **SPEED** *var* in EDIT mode. Displayed as a percentage of the maximum speed setting.	Defined by command **SPEED** *var* in DIRECT mode. Displayed as a percentage of the maximum speed setting.
	Defined by teach pendant command **SPEED** in Joint mode. (Entered as a percentage of maximum joint speed.) Displayed as a percentage of the maximum joint speed setting.

Note: Refer to the SPEED and SPEEDL commands.

SPEED DIRECT/EDIT

Format: `SPEED[A/B]` *var*

`SPEEDC` *var axis*

Where: *var* is a variable or constant.

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

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

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

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

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

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

Examples:

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

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

DIRECT/EDIT SPEEDL

Format: `SPEEDL` *`var`*

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

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

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

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

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

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

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

Examples:

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

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

SPLINE / SPLINED DIRECT/EDIT

Format:

`SPLINE` *`pvect n1 n2`* [*`duration`*]

`SPLINED` *`pvect n1 n2`* [*`duration`*] EDIT mode only

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

Description:

`SPLINE` Moves the axes through or near any number of consecutive vector positions, from *n1* to *n2*, without pausing, in a smooth and continuous movement.

`SPLINED` Same as SPLINE, except that the command following the SPLINED command will not begin execution until the robot has reached last position (as in MOVED command).

All positions in the vector must be the same type; either Absolute Joint or Absolute XYZ.

The SPLINE commands generate a smooth path of robot movement through or close to the points of the vector, from *n1* to *n2*. The trajectory is calculated so that the speed and acceleration are kept within safe limits, according to parameters 180+*axis* (maximum speed) and 520+*axis* (maximum acceleration). At low speeds, the trajectory passes through the positions in the vector. At high speeds, the trajectory "rounds the corners" in order to keep acceleration within safe limits.

The speed of the movement between any two consecutive positions is constant.

Duration is defined in hundredths of a second. Commands which do not include a *duration* argument are executed according to the SPEED or SPEEDL setting.

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

The trajectory of a SPLINE movement is determined by the type of the positions contained in the vector.

Joint SPLINE

The Joint SPLINE trajectory is used if the vector contains Absolute Joint positions.

If *duration* is not specified in the command, the movement is executed according to a preceding SPEED command.

The trajectory goes through the positions in a joint movement, (as in a MOVE command). The joint speed is kept constant during the movement, except for acceleration and deceleration at the start and end of the SPLINE movement.

This type of SPLINE gives the fastest movement possible.

The Joint SPLINE trajectory is most suitable for applications which require a smooth and quick path, such as pick and place operations, and palletizing.

Linear SPLINE

The Linear SPLINE trajectory is used if the vector contains Absolute XYZ positions.

The Linear SPLINE command is applicable only to robot (group A) axes.

If *duration* is not specified in the command, the movement is executed according to a preceding SPEEDL command.

The trajectory goes through the positions in a linear movement (as in a MOVEL command). The linear speed of the robot's TCP (tool center point) is kept constant, except for acceleration and deceleration at the start and end of the SPLINE movement.

The Linear SPLINE trajectory is most suitable for applications which require a geometrical path, such as welding, spray painting, gluing, and deburring.

STAT[US] DIRECT

Format: STAT

STATUS

Description: Displays the status of active user programs. The four columns provide the following information:

- Program name.
- Program priority.
- Current status of program.

  PEND is displayed if a program is waiting for a movement command to be completed.
- Program's current line number and the command being executed.

Example:

```
>STAT
job name    priority    status      position
BOOM           5        DELAY       3:DELAY
DEMO           5        PEND        15:MOVED
SS1   9     SUSPENDED   312:HERE
```

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.

Note: 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.

DIRECT TEACH

Format: TEACH *pos*

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

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

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

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

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

```
>TEACH PP
   X --[500.00] >
   Y --[0.00] >
   Z --[300.000] >
   P --[-0.900] >
   R --[0.00] >
```

The Cartesian (X, Y, Z) coordinates are defined in millimeters. Pitch and roll (P, R) values are defined in degrees.

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 (accurate to a micron or a millidegree.)

If the position entered is not valid, the coordinates are not accepted, and an error 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*]

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.

If *pos2* has an alphanumeric name, it must first be defined using the DEFP or DIMP command. The DEFP command is not required if the position is a *numerically* named position for group A; it will be automatically defined when entered as part of the command.

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

The Cartesian (X, Y, Z) coordinates are defined in millimeters. Pitch and roll (P, R) values are defined in degrees.

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 (accurate to a micron or a millidegree.)

Examples:

-
```
>TEACHR OVER
    X [.] > 0
    Y [.] > 0
    Z [.] > 60.5
    P [.] > 0
    R [.] > 0
```
Relative position OVER will always be 60.05 mm vertically above the current position of the robot.

-
```
>DEFP PLACE
>DEFP OVER
>HERE PLACE
>TEACHR OVER PLACE
    X [.] > 0
    Y [.] > 0
    Z [.] > 200
    P [.] > 0
    R [.] > 0
```
Positions PLACE and OVER are defined;

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

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.

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 immediately turns on the corresponding output. For example, if input 7 is on, output 7 turns on.

  Simulate the activation of an input by manually shorting an input with a wire:

  - If the input is configured as **sink**: short the input to COM+ of the same block.
  - If the input is configured as **source**: short the input to COM- of the same block.

The TEST procedure is in the RUN state until it you abort it.

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

[RUN] 999 [ENTER]

Format: `TON [n]`

`TOFF [n]`

Where: *n* is an axis, $0 \leq n \leq 12$

Description:

`TON [n]`	Switches ON the thermic motor protection for all axes, or for specific axis.
`TOFF [n]`	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.*

TOOL DIRECT/EDIT

Format: TOOL *length offset angle*

Where: *length* is the distance from flange to the TCP; defined in microns.
offset is the distance from the axis of symmetry of the flange to the TCP; defined in microns.
angle is the angle of TCP relative to the horizontal position when link 4 is horizontal (**ER IX** and **MK2**) and roll is 0 (all robots); defined in thousandths of a degree.

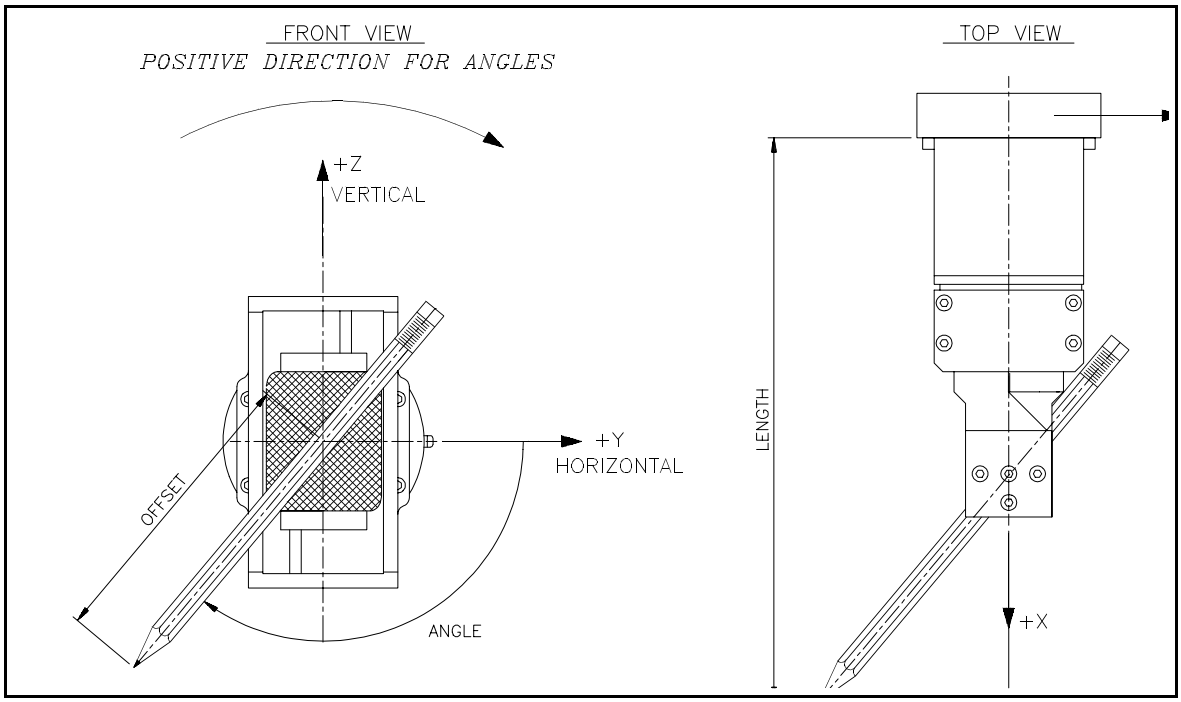
length, *offset* and *angle* can be a variable or a constant.

Description: TOOL defines the position of the end effector relative to the robot's flange. MOVEC, MOVEL, Linear SPLINE, and teach pendant movement commands are executed according to the robot's TCP (tool center point).

Example:

```
GLOBAL L O A
SET L=200000        Length is 200 mm.
SET O=75350         Offset is 75.35 mm.
SET A=45350         Angle is 45.3°
TOOL L O A
```

Note: The TOOL command sets the values of parameters 308, 309 and 310. When the parameter values are defined by the user, default settings for TOOL can be loaded from a parameter backup file.



EDIT TRIGGER

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, $1 \le n \le 16$;
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 an input state (off or on) is not specified, 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                    Program START activates program DRILL for the
SET OUT[3]=1                first time; thereafter, the TRIGGER command
DELAY 500                   within program DRILL reactivates program DRILL
SET OUT[3]=0                whenever input 15 is turned on.
MOVE P27
TRIGGER DRILL BY IN 15 1
END

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

UNDEF DIRECT/EDIT

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 coordinate values of position 5 in vector VECTV.
- `UNDEF VECTV` Clears the coordinate values of all positions in vector VECTV.

Note: This command does not create a program line.

DIRECT VER

Format: VER

Description: Displays the EPROM version and creation date.

Example: ■ >**VER**

```
- - ESHED ROBOTEC - -
CONTROLLER: B
VERSION: F2.28.04
DATE: 07/02/94
```

WAIT EDIT

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

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

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

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

Examples:

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

DIRECT ZSET

Format: ZSET

Description: This command initializes the index pulse on the encoders by setting to zero the value of all parameters in the range 420+*axis*.

The command should be executed before homing the axes for the first time, or after maintenance of encoders or mechanical components.

* 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 @ command 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 commmand 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 TCP is controlled.

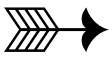
When Manual Keyboard mode is active, use the following keys to change the movement coordinate sysetms:

J	Joints coordinate system
X	Cartesian (XYZ) coordinate system.

The following chart summarized the resulting movements when the axes are controlled from the keyboard in Manual mode. The axes will move as long as the activating key is depressed, or until a fixed stop is reached.

Note the differences in axis action for different robots and different coordinate systems.

	SCORBOT-ER IX, PERFORMER-MK2 Vertical Articulated Robot		**SCORA-ER 14 Horizontal Articulated Robot**	
Keys	**ACTION XYZ mode**	**ACTION JOINT mode**	**ACTION XYZ Mode**	**ACTION JOINT mode**
X / J	Toggles between JOINTS and XYZ mode.			
1 / Q	All/some axes move in order to move TCP along X+ and X– axis.	Moves BASE (axis 1) counterclockwise and clockwise.	All/some axes move in order to move TCP along X+ and X– axis.	Moves axis 1 counterclockwise and clockwise.
2 / W	All/some axes move in order to move TCP along Y+ and Y– axis.	Moves SHOULDER (axis 2) up and down.	All/some axes move in order to move TCP along Y+ and Y– axis.	Moves axis 2 counterclockwise and clockwise.
3 / E	All/some axes move in order to move TCP along Z+ and Z– axis.	Moves ELBOW (axis 3) up and down.	Moves axis 3 (Z-axis) up and down.	
4 / F	Shoulder, elbow and pitch axes move, causing the pitch angle to change while maintaining the position of the TCP.	Moves wrist PITCH (axis 4) up and down.	Moves axis 4 (roll) counterclockwise and clockwise.	
5 / T	Moves wrist ROLL (axis 5) clockwise and counterclockwise (as seen from above, when end effector pointed down).		Moves axis 5, but not gripper.	
6 / Y	Moves axis 6, but not gripper.			
7 / U	Moves axis 7.			
8 / I	Moves axis 8.			
9 / O	Moves axis 9.			
0 / P	Moves axis 10.			
- / [	Moves axis 11.			
= /]	Moves axis 12.			



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.

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-B** has five reserved names for user programs: AUTO, BACKG, CRASH, EMERG and START. You can 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 or reset.

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.

BACKG

The BACKG program is automatically executed when the controller is powered on or reset, and as soon as the EMERGENCY button is released.

The BACKG program is a protection routine which can serve to prevent unintentional user errors which could result in physical injury or damage to the robotic system. BACKG continually checks the safety of operations and responds to hazardous situations.

BACKG can be written to suit the specific requirements of the user's application. For example, it can check and ensure that the robot's base axis remains stationary while the gripper is placing an object into a machine. Thus, if a command entered from the keyboard or teach pendant causes the base axis to move, BACKG can immediately issue a COFF command to halt the movement of the robot.

BACKG runs continually in the background and *will not be aborted* by any of the following:

- Entering the command A or <Ctrl>+A.
- Switching control to the teach pendant.
- Executing HOME, HOME *axis* or HHOME *axis*.

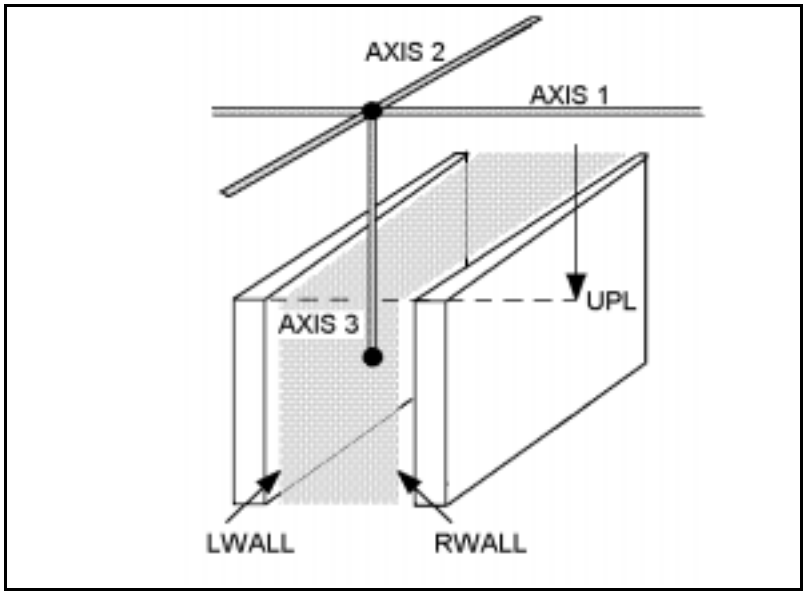
Since BACKG cannot be aborted as easily as other **ACL** programs, you must be absolutely certain that BACKG will not result in a dangerous situation, such as unexpected movement of a robot or device.

To abort the running of BACKG, do one of the following:

- Enter the command A BACKG.
- Use the STOP BACKG command line in another program.
- Press the EMERGENCY button.
  Remember! Releasing the EMERGENCY button automatically activates BACKG.

Example

The following sample program ensures that a gantry robot travels between two walls of a given, equal height, without making contact with the walls themselves, as indicated by the shaded area in the diagram below.



```
   PROGRAM BACKG
   *****************
LABEL 1
IF ANOUT[1]<>0              If ANOUT does not equal 0, the axis
   ANDIF ANOUT[2]<>0        group is not in COFF mode.
   ANDIF ANOUT[3]<>0
IF ENC[3]<UPL               If axis 3 is lower than wall height, and
   IF ENC[1]>LWALL             if axis 3 is too close to left wall, or
      ORIF ENC[1]<RWALL        if axis 3 is too close to right wall,
      FOR I=1 TO 3
          CLRBUF            Stop and disable all axes.
      ENDFOR
      @COFF
   ENDIF
ENDIF
DELAY     5                 Check again, every 50 ms.
GOTO      1
END
```

CRASH

The CRASH program is automatically executed when an impact, thermic, or "excessive 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
```

EMERG

The EMERG program is automatically executed when any emergency switch or button is pressed.

You may want to create this program in order to turn inputs and outputs off or on when the emergency status is in effect.

Example

```
   PROGRAM EMERG
   *****************
* TURNS OFF OUTPUT FOR SAFETY
SET OUT[7]=0
END
```

START

The START program is automatically executed when the Start push button on the auxiliary user control box is pressed.

Be sure the Auxiliary User Control Box is properly connected to the controller and parameter 16 defines the input to which the Start switch is connected.

This program can be used to start a process manually and immediately, by simply pressing a button.

Example

```
   PROGRAM START
   *****************
* INPUT 10 IS SELECTOR
IF INPUT[10]=1
   GOSUB PROG1
ELSE
   GOSUB PROG2
ENDIF
END
```

As soon as the Start button is pressed, the status of input 10 is checked. If the input is on, PROG1 is executed; if the input is off, PROG2 is executed.

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 0.5`
- `SETPV POSITION 1 80000`
- `SETPVC POSITION Y 5000`

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-B** contains 14 system variables:

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

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.

The values of IN, ENC, HS, ZP, CPOS, TIME, LTA, LTB, and MFLAG are updated at every controller clock tick. 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`		

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.

HS[*n*]

The value of this variable indicates the status of the home switch for the specified axis at its current position.

This is 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.

Example

Purpose	Program Command	Notes

Used during maintenance, repair and testing of the controller.	`LABEL 1`
	`PRINTLN HS[7]`
	`DELAY 100`
	`GOTO 1`
		Depending on the actual wiring connections, the value of HS will change to either 1 or 0 when the home switch is detected.

ZP[*n*]

The value of this variable indicates the status of the encoder index pulse for the specified axis at its current position.

This is 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.

Example

Purpose	Program Command	Display	Notes
Used during maintenance, repair and testing of the controller.	PROGRAM ZP ********** LABEL 1 VAR=ZP[1] IF VAR=1 PRINTLN VAR STOP END IF GOTO 1 PROGRAM MOVE ********** SPEED 1 LABEL 1 MOVE POS1 MOVE POS2 GOTO 1	Unlike HS, the value of ZP is always 0, and switches to 1 when the index pulse is detected.	Simultaneously run programs ZP and MOVE. Or, activate program ZP, and then move the axes by means of the teach pendant at the slowest speed setting.

CPOS[*n*]

This variable contains the current target position in encoder counts. At each clock tick this value is calculated by the controller for the specified axis.

This is 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.

Example

Purpose	Program Command	Display	Notes
To determine the position error of the axis.	SET ERR=CPOS[1]-ENC[1] PRINTLN ERR	82	The values of ENC and CPOS are compared. Position error of axis 1 = 82 encoder counts.

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 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, the 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

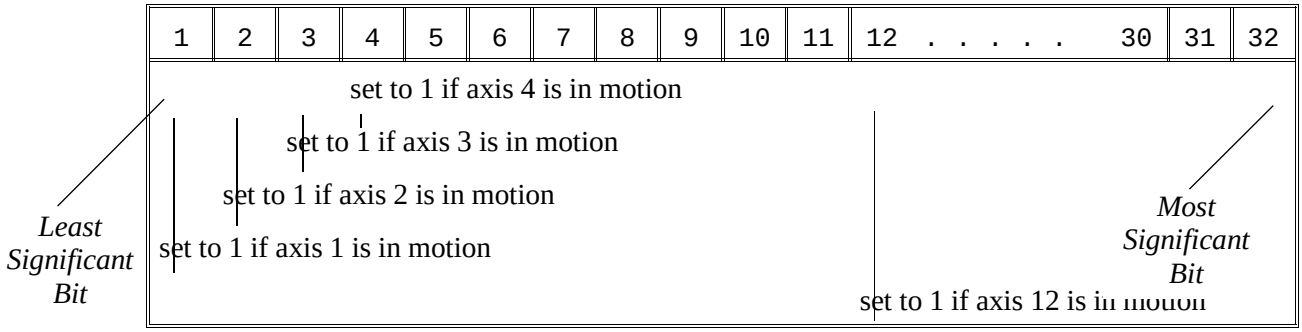
Purpose	Program Command
To synchronize the arrival of group A axes and group B axes at their respective destinations. (POSA5 is a position of group A and POSB3 is a position of group B.)	PROGRAM SYNCH ********** MOVE POSA5 SET V=LTA-TIME MOVE POSB3 V

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:



Bits 13 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	2048
Group	Group A					Gripper	Group B		Group C			
Axis	1	2	3	4	5	6	7	8	9	10	11	12

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, ERRPR and ERRLI

When a system error occurs during run time, these error variables are assigned values in the following manner:

- The value of ERROR indicates the specific error. An error message is also displayed.
- The value of ERRPG indicates the identity of the program (ID number, as displayed by DIR command) that initiated the task in which the error occurred.
- The value of ERRLI indicates the line number within the program at which the error occurred.

Refer to Chapter 5 for explanations of the error messages.

These are read/write variables. Each of these three variables *must have an initial value of* 0; otherwise, the value of the variable will not change during program execution.

Example

Purpose	Program Command	Display	Notes
To identify run-time errors.	PROGRAM MOVE ********** LABEL 1 MOVE POS66 MOVE POS67 GO TO 1		Simultaneously run programs ERROR and MOVE.
	PROGRAM ERROR ********** SET ERROR=0 SET ERRPR=0 SET ERRLI=0 * wait for error to occur WAIT ERROR <>0		
	PRINTLN "ERROR NO. " ERROR	53	Error ID number
	PRINTLN "TASK ID " ERRPG	7	Task number
	PRINTLN "AT LINE " ERRLI	144	Line number

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]`		

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.

In PRIVILEGE mode ANOUT[*n*] is a read/write variable; otherwise, it is 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.

The DAC value is in the range: ±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. (PRIVILEGE mode only)

CHAPTER 5

System Messages

This chapter contains a listing of the system messages which may appear on your screen during **ACL** operation and programming.

Italicized text in the messages shown here will be replaced by the actual item (for example, name of program, name of position, axis number) when the message is displayed.

When an equivalent message is also displayed on the teach pendant, that message is included [in brackets] in the description. Refer to the *Teach Pendant for Controller-B User's Manual* for an alphabetical listing and of the teach pendant messages.

The explanations of error messages include instructions for correcting the situation which caused them. System messages which prompt the user for a Yes/No response and other self-explanatory messages are not not included in this chapter.

When a system error occurs at run time, system variables ERROR, ERRPG and ERRLI receive values which indicate, respectively, the type of error, the program, and the line number at which the error occurred. Each of these three variables *must have an initial value of* 0; otherwise, the value of the variable will not change during program execution. Refer to the description of the error variables in Chapter 4.

(3) `ACL - Unknown command.`

The command entered is not a recognized ACL command.

(4) `SHOW - Unknown command.`

The command entered after the SHOW request is missing or is not a recognized SHOW command.

(6) *`Prg_name`* `not found.`

A program with the specified name was not found, or does not exist.

(12) `Cannot remove files while jobs are running.`

You attempted to delete a program (REMOVE) while that program or another program was running.

- You must first stop execution of all programs. You may then enter the REMOVE command.

(17) `Program` *`prog`* `is running.`

When in DIRECT mode, you cannot RUN a program while it is already running.

(18) `Program` *`prog`* `is running. Cannot access editor.`

You attempted to EDIT a program while it is running.

- You must stop the program before you can edit it.

(22) `Cannot delete constant or system variables.`

System variables and constants cannot be deleted.

(24) `TP Teach mode enabled. All programs aborted.`

The Auto/Teach switch on the hand-held teach pendant has been switched from Auto to Teach. For safety, all programs and movements are aborted.

(33) `*** UPPER LIMIT AXIS` *`n`*

During keyboard or TP manual movement of the specified axis, its encoder attained its maximum allowed value.

- Move the axis in the opposite direction.

(34) `*** LOWER LIMIT AXIS` *`n.`*

During keyboard or TP manual movement of the specified axis, its encoder attained its minimum allowed value.

- Move the axis in the opposite direction.

(35) `Input or Output must be disabled prior to FORCE operation.`

A FORCE command appears without a prior DISABLE command.

- Before you can force an input to a value, the input must be in the DISABLE mode.

(36) `Invalid index or value.`

[Index or data error.]

The command has an incorrect index value.

(39) `Incorrect number of arguments.`

The command has an incorrect number of arguments.

(40) Cannot hard home *axis n*.

[No hard Homing]

The specified axis has not been configured for hard homing.

- Use the HOME command (instead of HHOME). OR
- Check the type of homing suitable for that axis. If necessary, change the system parameters to allow hard homing of the axis.

(48) HELP - Unknown command.

The command entered after the HELP request is missing or is not a recognized HELP command.

(51) *** HOME FAILURE AXIS *n*.

[HOME FAIL *n*.]

The homing procedure failed for the specified axis. Possible causes:

(1) The home microswitch was not found.

(2) The motor power supply is switched off.

(3) Hardware fault on this axis.

(52) *** TRAJECTORY ERROR !

This error affects the ERROR system variable.

During movement, the robot arm reached its envelope limits, and the system aborted the movement. This may occur when executing the following types of movements: linear (MOVEL), circular (MOVEC) , MOVES, and SPLINE. Since the trajectory is not computed prior to motion, the movement may exceed the limits of the working envelope.

- Modify the coordinate values of the positions which define the trajectory.

(53) *** IMPACT PROTECTION axis *n*

[IMPACT AX *n*]

This error affects the ERROR system variable.

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) The motor power switch is turned off.
(4) An encoder fault.
(5) A mechanical fault.
(6) 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*

[RANGE *n*]

This error affects the ERROR system variable.

An attempt was made to record a position (HERE, HEREC, etc.) 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*

[THERMIC AX *n*]

This error affects the ERROR system variable.

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).

(58) Cannot execute program. Waiting for an available task...

Too many tasks are running concurrently, and you cannot run another one. Possible causes:

(1) You have run too many programs concurrently, and have not terminated them.

(2) One program runs indefinitely, and concurrently to itself, without being terminated.

(3) One of the programs executes another program in a closed loop.

- Abort all programs. A list of the aborted programs will be displayed on the screen. This will indicate which program is running too many time. OR
- Use the STOP command . This will halt all concurrent executions of the same program. OR
- Correct your program(s) as necessary.

(61) `*** RUN TIME WARNING *** Triggered program not found.`

An attempt was made to trigger a program which does not exist.

(62) `*** RUN TIME WARNING *** Invalid I/O number for trigger.`

The input or output number used in the TRIGGER command is out of range.

(63) `*** WARNING ***` *`var`* `already exists.`

You attempted to define a variable with a name which is already in use.

(64) `Not enough contiguous memory for array.`

The controller does not have enough memory available for the array you want to define.

- Backup and restore all your programs. This will rearrange the memory allocation. OR
- Type PURGE on keyboard; all unused variables will be deleted. OR
- Backup your current programs. Use the CONFIG command to increase the number of positions and/or variables in the configuration. Then restore all your programs.

(69) `TP Teach mode disabled. Type AUTO <Enter> for Auto mode.`

The Auto/Teach switch on the teach pendant has been switched from Teach to Auto. The teach pendant is no longer operative.

- Enter the command AUTO to reestablish keyboard control.

(72) `CONTROL DISABLED.`

[CONTROL DISABLED.]

Motors have been disconnected from servo control. Possible causes:

(1) COFF (control off) command was issued.

(2) CON (control on) has not been issued; the motors have not been activated.

(3) A previous error (such as Impact Protection, Thermic Protection of Trajectory Error) activated COFF, thereby disabling the arm.

- If the axes were disabled due to Impact, Thermic, or Trajectory error, check the last movements executed. A movement may have failed because excessive speed or an invalid position resulted in a trajectory beyond the limits of the robot envelope.

(73) `CONTROL ENABLED.`

[CONTROL ENABLED.]

Motors are now under servo control and can be activated .

(74) `No coordinate values for position.`

[Point not assigned.]

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.

(75) `Invalid position coordinate values.`

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

(76) `*** WARNING *** No robot configuration.`

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

(77) `*** WARNING *** Array` *`name`* `truncated.`

During a restore operation, the controller did not have enough memory to restore the specified array, and the array has been truncated.

- Backup and restore all your programs. This will rearrange the memory allocation. OR
- Type PURGE on keyboard; all unused variables will be deleted. OR
- Backup your current programs. Use the CONFIG command to increase the number of positions and/or variables in the configuration. Then restore all your programs.

(78) `Performing configuration. Please wait...`

After the configuration settings have been entered, the system is initialized.

(79)
```
USER RAM CHECKPOINT ERROR !
INSTALLING DEFAULT SETUP.
Reconfigure and reload parameters for specific robot used.
```

When the controller is powered on, battery backed up memory is checked at predefined location. If the values found at these locations have changed, the controller detects an error, and erases all memory. Default setup is then installed.

Possible cause: Low battery.

- Check the battery and housing. Replace the battery, if required. Reconfigure the controller, and restore all your programs from backup.

(85) `No available task. Cannot execute program.`

When in DIRECT mode, the program indicated by the RUN command cannot be executed since the task memory is full.

(86) `Home task is already active.`

You attempted to use the HOME command while a HOME command is still being executed.

- Wait until the current HOME command has been completed. OR
- Abort the homing by entering the command A <Enter>.

(89) `Cannot execute XYZ movement at that position.`

The movement could not be executed because the XYZ coordinate system is not supported in that part of the envelope.

- Switch to JOINT mode, and manually move to a valid XYZ position.

(92) `Current configuration includes auxiliary RS232 card. Press C <Enter> if immediate reconfiguration required.`

Your system is currently configured for an RS232 auxiliary card.

If the card exists, simply wait and the system will start normally.

If you have removed the auxiliary RS232 card, press C <Enter> to bring up the configuration menu, and reconfigure the controller.

If you do not change the configuration to indicate that the auxiliary RS232 card is not installed, the system will try to communicate with the card, resulting in a BUS ERROR.

(95) `INDEX pulse not found` *`n`*

The index pulse of the encoder was not found during the homing of the specified axis. Possible causes:

(1) The distance between the index pulse and the home switch transition position has changed, due to a mechanical fault on the axis or a maintenance procedure (such as replacement of the motor, motor belt, encoder, or gear).

- Enter the command ZSET. Then retry homing.

(2) Index pulse faulty.

- Check the encoder and wiring.

(98) `Axis` *`n`* `not configured for homing.`

The homing parameters for the axis (PAR 460+*axis* and PAR 600+*axis*) are set to 0; as a result, the homing procedure will not be performed on the axis.

(99) `*** SPEED TOO FAST axis n.`

[TOO FAST AX *n.*]

This error affects the ERROR system variable.

Possible causes:

(1) The controller has detected a movement which is too fast; that is, the required displacement of the encoder, as calculated from the speed limit parameter, PAR 180+*axis*, is too great.

(2) Since the trajectory is not calculated prior to a linear or circular movement, the linear or circular movement may cause one of the joints to move too fast.

- Lower the value of speed for that movement.

(100) `Out of memory - programs.`

The number of programs (created in EDIT) has reached the maximum limit allowed by the controller configuration.

- Delete unused programs. OR
- Backup your current programs. Use the CONFIG command to increase the number of programs in the configuration. Then restore all your programs.

(101) `Syntax error.`

The command has a syntax error.

(102) `Undefined variable(s).`

The variable has not been defined, or has an undefined variable index.

(103) `Missing argument.`

The command is missing an argument.

(104) `Undefined program.`

A program with the specified name does not exist.

- EXIT the program you are currently editing. Use EDIT to create a new program with the specified name (the program may remain empty). You may then resume editing your original program.

(105) `Undefined variable - PRIORITY.`

The variable in the PRIORITY command has not been defined, or has an undefined variable index.

(106) `Undefined variable - PEND/POST.`

A variable in the PEND/POST/QPEND/QPOST command has not been defined, or has an undefined variable index.

(109) `Out of memory - program lines.`

The number of program lines has reached the maximum limit allowed by the controller configuration. OR

- Delete unused programs.
- Backup your current programs. Use the CONFIG command to increase the number of program lines in the configuration. Then restore all your programs.

(110) `Position is undefined, not recorded, or for incompatible group.`

[POINT NOT FOUND]

The position you attempted to use cannot be reached. Possible causes:

(1) The position has not been defined.

(2) The position coordinate values have not been recorded.

(3) The command indicated the use of a position belonging to a group which is incompatible with the command.

- Define and/or record the position.

(111) `Out of memory - variables.`

The number of variables has reached the maximum limit allowed by the controller configuration.

- Type PURGE on keyboard; all unused variables will be deleted. OR
- Backup your current programs. Use the CONFIG command to increase the number of variables in the configuration. Then restore all your programs.

(112) `Out of memory - group` *`group`* `positions.`

The number of positions for the specified group has reached the maximum limit allowed by the controller configuration.

- Backup your current programs. Use the CONFIG command to increase the number of positions for the specified group. Then restore all your programs.

(113) `Invalid position name.`

The position could not be defined because the name is invalid.

(114) `First variable undefined.`

The first variable in the command has not been defined, or has an undefined variable index.

(115) `Second variable undefined.`

The second variable in the command has not been defined, or has an undefined variable index.

(116) `Undefined variable - SPEED.`

The variable in the SPEED command has not been defined, or has an undefined variable index.

(117) `Out of memory - strings.`

The number of string commands (PRINT, * and @) has reached the maximum limit allowed by the controller configuration.

- Delete unused strings. OR
- Backup your current programs. Use the CONFIG command to increase the number of strings in the configuration. Then restore all your programs.

(118) `Invalid argument(s).`

The command has an invalid argument.

(119) `Undefined variable (loop counter) - FOR.`

The variable which sets the loop counter in the FOR command has not been defined, or has an undefined variable index.

(120) `Invalid variable (loop counter) - FOR.`

You cannot use a constant value for the loop counter in the FOR command.

(121) `Third variable undefined.`

The third variable in the command has not been defined, or has an undefined variable index.

(122) `Undefined variable - TRIGGER.`

The variable which sets the input/output in the TRIGGER command has not been defined.

(123) `Invalid input/output.`

The input or output number used in the command is out of range.

(124) `Invalid or undefined axis.`

The variable used to designate the axis has not been defined, or its index has not been defined, or the axis is not configured (out of range).

(125) `Name already in use or invalid axis.`

(1) You attempted to define a position (DEFP) with a name which is already in use.

(2) You used at invalid axis number when attempting to define a position in group C.

(126) `Invalid dimension.`

The dimension of the array (DIM, DIMG or DIMP) must be designated by a constant value.

(127) `Syntax error - index`

Index brackets [ ] are not balanced.

(128) `Constant too big.`

ACL's limit for a constant absolute value is 999999999.

(129) `Invalid sequence in program` *`prog`*`.`

Program contains a logic error. For example:
(1) FOR loop not closed by an ENDFOR.
(2) IF section not closed by an ENDIF.
(3) ELSE section not closed by an ENDIF.

(130) `Nesting too deep at line` *`n`*`.`

Nesting is too deep in FOR/ENDFOR or IF/ENDIF routines.

(131) `Missing IF for line` *`n`*`.`

An ENDIF command appears without a preceding IF command.

(132) `Missing IF for line` *`n`*`.`

An ANDIF or an ORIF command appears without an immediately preceding IF command.

(133) `Missing FOR for line` *`n`*`.`

An ENDFOR command appears without a preceding FOR command.

(135) `Label number already defined.`

You attempted to use a LABEL *n* command which is already in use in the same program.

(136) `Missing or invalid operator.`

An operator in the command is either missing or invalid.

(137) `Missing argument - SPEED.`

The SPEED command is missing the speed argument.

(138) `Name already in use.`

You attempted to define a variable or position array (DIM, DIMG or DIMP) with a name which is already in use.

(139) `Relative chain loops or exceeds 32 positions.`

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.

(140) `Invalid variable name - too long.`

You attempted to define a variable with a name which exceeds 5 characters.

(142) `Missing LABEL for GOTO at line` *`n`*.

A GOTO command appears without a corresponding LABEL in the same program.

(143) `Error in downloaded command, at line` *`n`*.

An error occured while restoring a program. Possible causes:

(1) The backup file is corrupted.

(2) The controller configuration has changed, and a command in the restored program is not valid for the current configuration.

(144) `Invalid argument - too long.`

The command has an argument which exceeds 10 characters.

(145) `Position is not defined or is not a vector.`

[Pnt not def or array.]

The position has not been defined, or the name used does not define a vector.

(146) `***WARNING***` *`name`*`: Invalid name.`

You cannot use a constant in the DEFINE or GLOBAL command.

(147) `Invalid port number.`

The port number used in the command is out of range.

(148) `To perform` *`action`* `- release emergency button.`

[EMERGENCY]

The emergency switch has been pressed.

- To resume normal operation: releasae the emergency button; abort the user routine, EMERG, if it exists; then activate CON (or HOME).

(149) `Cannot change parameter - privilege.`

Most system parameters are password-protected, and can only be altered when the PRIVILEGE mode is active.

(150) `Command ignored - privilege.`

You attempted to execute a password-protected command. The command can only be performed when the PRIVILEGE mode is active.

(152) `Invalid inputs for control box (parameters 15/16).`

You attempted to use an input number which is out of range, incompatible, or already in use.

(153) `Cannot execute command. TP must be in Auto mode.`

This command can only be executed when the TP is in Auto mode.

(154) `Value too low - SPEED.`

The value for the SPEED command must be greater than 0.

(156) `Excessive linear speed.`

The value entered for the linear speed (SPEEDL) exceeds the maximum limit, as defined by parameter 536.

(157) `Excessive speed required.`

The value entered for the time argument of the MOVE, MOVEL, MOVEC, MOVES, or SPLINE command would cause the speed of movement to exceed the maximum limit.

(158) `Excessive speed required or positions too close.`

During movement in a SPLINE trajectory, more than one point will be reached within one clock tick (8.3 or 10 millisecond). The SPLINE movement cannot be completed.

- Increase the distance between points, or use a slower speed.

(159) `No free space to insert position.`

[No free space.]

All positions in the vector have coordinate values; no memory available.

- You must first delete (DELETE) any unrequired positions in the vector.

(160) `Insert position is empty, use HERE or HEREC.`

[use HERE or HEREC.]

The vector position at the place of insertion does not have coordinate values. The INSERT command cannot be used. Use HERE or HEREC instead.

(161) `INSERT allowed only if position name begins with &.`

The INSERT/DELETE operations are available only for vectors whose names were defined with the prefix &; for example, `&PVEC[20]`.

(162) `INSERT/DELETE not allowed for a single axis group.`

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

(163) `Index value - out of range.`

[INDEX RANGE]

The value of the index is out of the defined range.

(164) `DELETE allowed only if position name begins with &.`

The INSERT/DELETE operations are available only for vectors whose names were defined with the prefix &; for example, `&PVEC[20]`.

(165) `Cannot execute command when in TP Teach mode.`

Switch the Auto/Manual switch on the teach pendant to Auto.
Then enter the command AUTO from the keyboard.

(201) `Motor power switch is OFF.`

[MOTORS OFF]

The command could not be executed because the motor power supply is switched off.

- Be sure the motor power switch is turned on. Activate CON.
  Then repeat the command.

(205) `PROGRAMS ABORTED.`

[All programs aborted]

Possible causes:

(1) The command A or <Ctrl>+A was entered from the keyboard.

(2) The Abort key on the teach pendant was pressed.

(3) The hand-held teach pendant was switched from Auto to Teach mode during program execution.

(4) The mounted teach pendant, in Teach mode, was removed from the fixture during program execution.

(206) `*** WARNING ***`
`'BACKG' is a reserved name for Background routine.`
`'BACKG' remains active unless EMERGENCY pressed.`
`'BACKG' cannot be aborted by A<Enter>.`

This message will appear when you begin to edit a user program with the reserved name BACKG. Refer to the section describing BACKG in Chapter 4, and heed all warnings given there.

(301) `Group/Axis has not been homed.`

[HOME NOT DONE]

You attempted to move the arm to a recorded positions, or to record a position, before homing was performed on the group or axis.

(302) `Arithmetic overflow (or division by zero)`

[ARITHMETIC OVFL]

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

(303) `No coordinate values for position` *n*.

[NO POSITION]

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.`

[AXIS DISABLED]

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) `Nesting too deep.`

[TOO DEEP NESTING]

Too many GOSUB subroutines are nested within one another.

(306) `Invalid program.`

[INVALID PROGRAM]

The RUN, GOSUB, TRIGGER command cannot be executed, due to faulty syntax or logic in the program.

(309) `Index value out of range.`

[INDEX RANGE]

The value of the index is out of range.

(310) `Invalid axis.`

[BAD AXIS]

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

(311) `Relative chain loops or exceeds 32 positions.`

[POINT LOOP]

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) `Invalid position coordinate values.`

[BAD POINT *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.

You attempted to use an XYZ movement command (MOVEL, MOVEC and Linear SPLINE) in which movement crosses from World Space A to B, or B to A.

- All positions referenced in the XYZ command must belong to *either* World Space A or World Space B.

(313) `Incompatible position type` *`pos`*.

[POINT TYPE *pos*]

You have attempted to use a position whose type is incompatible with the command; the error may be, for example:

(1) SHIFT of a relative position by XYZ coordinate values.
(2) SHIFTC of a relative position by JOINT coordinate values.
(3) SETPV of a relative position by XYZ coordinate values.
(4) SETPVC of a relative position by JOINT coordinate values.
(5) SHIFTC of a relative position to current position.
(6) SPLINE on a relative position.
(7) SPLINE on positions of differents type (all positions of SPLINE must be the same type).

(315) `Incompatible positions.`

[INCOMPATIBLE PNT]

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.`

[NO GRIPPER]

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) `Invalid Cartesian position` *`pos`*`.`

[BAD XYZ POSITION]

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

- Switch to JOINT mode.

(319) `Motor power switch is OFF.`

[MOTORS OFF]

The command could not be executed because the motor power supply is switched off.

- Be sure the motor power switch is turned on. Activate CON. Then repeat the command.

(320) `Robot is not in XYZ space.`

[NOT XYZ SPACE]

The robot is at a position at which the XYZ coordinate system is not supported.

- Switch to JOINT mode.

(321) `MOVES/SPLINE not allows for a single axis group.`

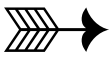
[BAD GROUP]

MOVES and Joint SPLINE can only be applied to a robot or multi-axis device which is dedicated to group A or group B. The command cannot be executed on a single axis group.

(322) `Position must have Absolute Joint coordinates.`

[NOT ABS POINT]

MOVES can only be executed on positions with absolute JOINT values.



CHAPTER 6

User Memory Configuration

The standard **ACL Controller-B** has 512KB of battery-backed CMOS RAM, which is allotted to both the system and the user.

For example, when the configuration is performed by means of the **ATS** hot-key (Alt+F1) combination, and the options SCORBOT-ER IX and 8 AXES are entered, the default allocation will be as follows:

400	Programs
7000	Program lines
3500	Variables
4400	Group A positions
4400	Group B positions
0	Group C positions
800	Comments

Only axis control Groups A and B are defined in the controller's default onfiguration. Group A is defined as axes 1 through 5 (for **SCORBOT-ER IX** and **PERFORMER-MK2**) or as axes 1–4 (for **SCORA-ER 14**). The electric gripper is defined as the axis following Group A: either axis 6, or axis 5. All remaining axes are defined as Group B. To define Group C axes, the **ACL** command CONFIG must be used.

These figures are calculated according to the specific amount of memory required by each data 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) $\times$ 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	$= 100 \times 11 = 1100$ bytes
550 Lines	$= 550 \times 10 = 5500$ bytes
200 Variables	$= 200 \times 16 = 3200$ bytes
450 Positions for group A	$= 450 \times 20 = 9000$ bytes
450 Positions for group B	$= 450 \times 18 = 8100$ bytes
150 Positions for group C	$= 150 \times 14 = 2100$ bytes
100 Strings/comments	$= 100 \times 12 = 1200$ 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 $32760 < 131072$; $(131072 = 128 \times 1024)$.

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.

Parameter Protection

The parameters are divided into two access groups: password-protected parameters and non-protected parameters.

Protected parameters may be accessed and manipulated only after you have activated the PRIVILEGE mode. This feature prevents accidental or incorrect manipulation of servo and other critical parameters.

The status of parameter 19 indicates whether or not the PRIVILEGE mode is active. If PAR 19=0, a password is required to access the protected parameters; if PAR 19=1, no password is required.

Non-protected parameters may be accessed and manipulated at any time. The following parameters do not required the PRIVILEGE mode:

- Gripper parameters: 73, 74, 75, 76, 274 and 275
- Smoothing parameter: 219
- Trajectory parameters: 220 and 236
- Position Error parameters: 261–272

Refer to the commands PASSWORD and PRIV[ILEGE] in Chapter 3 for more information on the PRIVILEGE mode.

Parameter Commands

The parameters may be accessed and manipulated by the following **ACL** 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 parameters 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.

Parameter Descriptions

ACL 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 set DAC (analog output) values are defined in the range ±5000 (equivalent to ± 24V on motor). Other parameter values are expressed in units such as encoder counts, controller clock ticks, linear measurements, and so on.

The parameters supplied with the robots 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.

The following two tables describe all the parameters for **Controller-B**.

- **Parameter Table 1** gives brief descriptions of the parameters, classified according to their functions.
- **Parameter Table 2** gives complete descriptions of the parameters, arranged in numerical order.

Non-protected parameters are indicated by shaded numbers.

Parameter Table 1		
System Setting Parameters		
I/Os for Auxiliary Control Box	15	Defines controller input # for Run/Hold Switch.
	16	Defines controller input # for Start Program Switch.
	18	Identifies the production model of the controller.
	19	Defines whether PRIVILEGE mechanism is required.
Axis Servo Control Parameters		
Basic Servo	20 + *axis*	Proportional feedback constants, K_p.
	40 + *axis*	Differential feedback constants, K_v.
	60 + *axis*	Integral feedback constants, K_i.
	360 + *axis*	Maximum DAC value for integral feedback.
	380 + *axis*	Average value of ratio: *analog output ÷ encoder counts*
Driver Gain	320 + *axis*	Gain of the PWM preamplifier of the driver card.
	340 + *axis*	Gain for tachometer feedback.
	340 360	*Reserved. Not for user manipulation.*
Global Servo Control Parameters		
	20	Defines servo control cycle time.
	78	Defines whether proportional and differential feedback constants are doubled at end of motion.
	79	Value by which integral feedback constants are divided during movement.
Axis Limit Parameters		
	100 + *axis*	Upper limit of axis motion, in encoder counts.
	120 + *axis*	Lower limit of axis motion, in encoder counts.
	540 + *axis*	Maximum encoder range.
Thermic Protection Parameters		
	140 + *axis*	Motor voltage/speed characteristics of a free running motor.
	160 + *axis*	Maximum DAC value that can be applied to a stalled motor.
Impact Parameters		
	240	Defines axis group which responds when impact detected.
Run Mode	240 + *axis*	Value for detecting homing start error.
	680 + *axis*	Value for detecting servo error at run time/during homing.

Parameter Table 1		
Manual Mode	280 + *axis*	Maximum torque allowed while moving the axes manually (TP or keyboard).
	780 + *axis*	Value for detecting servo error during manual movement.
	700 + *axis* 720 + *axis* 740 + *axis* 760 + *axis*	Values for setting manual movement torque limitation.
	80 + *axis*	Driver offset compensation. DAC value used to calculate torque.
Speed Setting Parameters		
	180 + *axis*	Maximum speed setting, in (degrees/second), (mm/second), or (encoder counts/10 ms).
	533	Maximum linear acceleration.
	534	Maximum pitch/roll acceleration.
	535	Maximum roll acceleration.
	536	Maximum linear speed.
	537	Maximum pitch/roll speed .
	538	Maximum roll speed.
	660 + *axis*	Maximum DAC value which can be applied to axis.
Manual Speed Parameters		
	220 + *axis*	Speed setting for manual operation.
	239	Acceleration at start of a manual movement.
	500 + *axis*	Deceleration of movement after key is released.
	238	Deceleration of Cartesian movement after key is released.
	294	Maximum speed for manual Cartesian movement.
Keyboard Manual Parameter		
	300	Time required for key repetition, to produce a smooth, continuous axis movement.
Homing Parameters		
Home Switch Search	200	Defines whether double homing procedure is used.
	200 + *axis*	Maximum DAC value allowed while homing.
	460 + *axis*	Speed of search for the home switch.
	560 + *axis*	Defines home switch polarity.
	13	Slows motion during home switch homing.

Parameter Table 1		
Index Pulse Search	400 + *axis*	Speed of search for the index pulse.
	420 + *axis*	Index pulse position.
	440 + *axis*	Number of encoder counts for a 90° turn of motor.
Hard Home Search	600 + *axis*	DAC value to be applied to the axis during search for hard home.
Cartesian Calculations Parameters These parameters define the mechanical arm lengths, encoder and gear ratios, and the robot's home position; they are used to calculate the Cartesian position of the arm. The robot's structural type must be correctly defined in the controller configuration.		
Rotation Scaling (for robot axes only)	33 34 35 36 37	Number of encoder counts for +90° of axis 1. Number of encoder counts for +90° of axis 2. Number of encoder counts for +90° of axis 3. Number of encoder counts for +90° of axis 4. Number of encoder counts for +90° of axis 5.
Horizontal Reference Position	1 2 3 4 5	Encoder 1 reading at horizontal reference position. Encoder 2 reading at horizontal reference position. Encoder 3 reading at horizontal reference position. Encoder 4 reading at horizontal reference position. Encoder 5 reading at horizontal reference position.
	260	Compensation of roll angle to maintain orientation of tool.
Length Parameters All units are in microns (10^{-3} millimeters) or millidegrees.		
	301	X coordinate of the rotation axis of arm link 2 when robot is at home.
	302	Y coordinate of the TCP when robot is at home.
	303	Z coordinate of the rotation axis of arm link 2.
	304	Length of link 2; from the first articulated joint.
	305	Length of link 3; from the second articulated joint.
	306	Distance from pitch axis / flange to TCP.
	307	Distance from pitch axis to flange.
	308	Length. Value from TOOL command.
	309	Offset. Value from TOOL command.
	310	Angle. Value from TOOL command.
	237	Cartesian limit.

Parameter Table 1		
Trajectory Parameters		
	219	Smoothing factor for MOVES movement.
	220	Sinusoid velocity profile for joint movement.
	236	Sinusoid velocity profile for linear trajectory.
	520 + *axis*	Maximum acceleration/deceleration.
	580 + *axis*	Deceleration smoothness.
	199	Rate of deceleration when Run/Hold switch is switched to Hold.
	260 + *axis*	Maximum position error at end of MOVED, MOVELD or MOVECD.
Gripper Parameters		
	73	Gripper encoder range.
	74	Encoder count at closed position.
	75	Maximum DAC value for closing and opening gripper.
	76	Duration of gripper closing and opening.
	274	Controller output # for pneumatic gripper.
	275	DAC value to be applied to the gripper after the completion of a CLOSE gripper command.
	276	DAC value to be applied at the start of gripper movement.
	277	Duration of PAR 276.
	260+*axis*	When *axis* =gripper: maximum fluctuation of encoder value while gripper blocked.

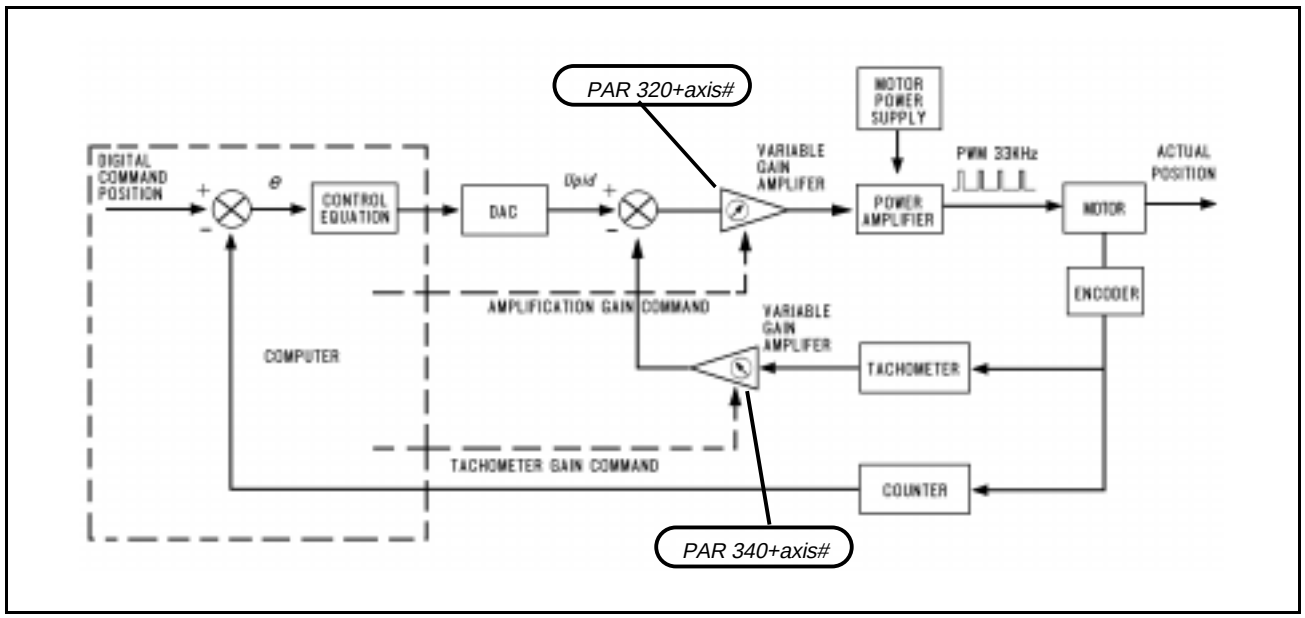
Parameter Table 2	
Parameter	**Description**
1 2 3 4 5	Used by controller for XYZ calculations. For robot axes only: Reading of encoder 1 at horizontal reference position. Reading of encoder 2 at horizontal reference position. Reading of encoder 3 at horizontal reference position. Reading of encoder 4 at horizontal reference position. Reading of encoder 5 at horizontal reference position. 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.
13	**Slow Homing**: Slows motion during home switch homing. This parameter slows the final search for the switch. If PAR 13=0: Default value If PAR 13=1: Twice as slow If PAR 13=3: Three times as slow
15	**Run/Hold program execution**. Defines the controller input to which the Run/Hold switch on the user (auxiliary) control box is connected. PAR 15 may have a value of 1–16. If PAR 15=0: switch not installed; not defined. PAR 199 defines rate of axis deceleration at switch to Hold.
16	**Start** program. Defines the controller input to which the START program button on the user (auxiliary) control box is connected. PAR 16 may have a value of 1–16. If PAR 16=0: button not installed; not defined.
18	**Controller ID**. Identifies the production model of the controller in which the software is operating. Factory set; *not* for user manipulation.
19	Determines whether **PRIVILEGE mode** is active or not. If PAR 19=0: Controller is always in PRIVILEGE mode; no parameter protection. If PAR 19=1: PRIVILEGE mode must be activated by means of password-protected PRIV command.
20	**Servo Cycle Time (Controller Clock Tick)**. Defines the servo control cycle time. Range: 30–200 tenths of a millesecond. For example: If PAR 20=100: a cycle is executed every 10ms. If PAR 20=60: a cycle is executed every 6ms.
20+*axis*	**Proportional** feedback constants, K_p, per axis.
33 34 35 36 37	Used by controller for XYZ calculations. For robot axes only: Number of encoder counts for +90° of axis 1. Number of encoder counts for +90° of axis 2. Number of encoder counts for +90° of axis 3. Number of encoder counts for +90° of axis 4. Number of encoder counts for +90° of axis 5.
40+*axis*	**Differential** feedback constants, K_v, per axis.

Parameter Table 2	
Parameter	**Description**
60+*axis*	**Integral** feedback constants, K_i, per axis.
73	**Gripper Encoder Range.** Number of encoder counts required to close gripper from fully open position. Valid only for a gripper with encoder feedback. If PAR 73=0, there is no encoder on the gripper.
74	**Gripper Encoder Count at closed position**. Valid only for a gripper with encoder feedback.
75	**DAC Value applied to gripper motor** when closing and opening gripper.
76	**Gripper Closing Duration**. The amount of time required to close and open gripper (in clock ticks). Valid only for a servo gripper *without* encoder feedback and for a pneumatic gripper.
78	**Servo Control**. If PAR 78≠0, the proportional (K_p) and differential (K_v) feedback constants are doubled at the end of motion.
79	**Integral Control**. Ratio of reduced integral feedback constants while axes are in motion. During movement the integral feedback constants are divided by PAR 79.
80+*axis*	**Driver Offset Compensation**. DAC value, used to calculate the torque. Range: 0–5000.
100+*axis*	**Upper Limit** of axis motion, in encoder counts, per axis.
120+*axis*	**Lower Limit** of axis motion, in encoder counts, per axis.
140+*axis*	**Motor Voltage/Speed Characteristics** of a free running motor. Needed for calculating the average thermic load of the motor. The current analog output value together with the motor characteristics result in a theoretical thermic analog output value, which is equivalent to the amount of power not translated into movement.
160+*axis*	**Thermic Protection**. Maximum DAC value that can be applied to a stalled motor. When the thermic error threshold is reached, power to a stalled motor is shut off. Range: 500–5000.
180+*axis*	**Maximum Speed Setting**. For robot axes: in (degrees/second). For Z-axis of **ER 14**: in (millimeters/second). For non-robot axes: in (encoder counts/10 milliseconds).
199	**Rate of Deceleration** of axis motion when the Run/Hold switch is switched to Hold. Defined as a percentage of maximum speed, as defined by PAR 180+*axis*: 1=slow; 100=immediate. Typical value: 4–5.
200	**Homing**. If PAR 200=0, the homing procedure will run twice for high precision. If PAR 200=1, a faster homing procedure will be performed. For robots with encoder index pulse (C-pulse), double homing is usually not required.
200+*axis*	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.

Parameter Table 2	
Parameter	**Description**
219	**Smoothing factor**. Defines the smoothness of MOVES movements. Range: 1–200. 1=no smoothing; 200=maximum smoothing.
220	Defines the shape of the **Sinusoid Velocity Profile for Joint (MOVE) Movement**. The value determines the percentage of motion time to be allocated for acceleration and deceleration when using the sinusoid motion profile.
220+*axis*	Defines the **speed setting for manual operation** of each axis. Defined as a percentage of the maximum speed, as defined by PAR 180+*axis*.
236	Defines the shape of the **Sinusoid Velocity Profile for Linear Trajectory**. The value determines the percentage of motion time to be allocated for acceleration and deceleration during linear (MOVEL) and circular (MOVEC) movements.
237	**Cartesian Limit** parameter. Minimum allowed angle, in degrees, between the two main robot joints while the robot is moving in Cartesian coordinates. This parameter prevents movement through the singular point of the Cartesian to Joint transformation.
238	**Manual Cartesian Movement Deceleration**. The rate of deceleration after a Cartesian manual movement key is released. A percentage of maximum acceleration, as defined by PAR 533, PAR 534, and PAR 536.
239	**Manual Movement Acceleration**. The rate of acceleration at the start of a manual movement. The value of this parameter is the number of clock ticks required for the arm to reach its requested normal speed.
240	**Impact Protection Response Mode**. Defines axis group which reponds when an impact condition is detected. If PAR 240≠0: when an impact error is detected, only the motors in the group to which the impacted motor belongs are shut off. If PAR 240=0: when an impact error is detected, all motors are stopped (default).
240+*axis*	**Homing Start Error**. If a position error (in encoder counts) at the start of the homing routine is greater than the parameter value, an error condition is detected.
260	**Roll Compensation for Maintaining Tool Orientation**. When executing a linear movement, if pitch value is close to +90° or -90°, the controller will compensate the roll angle in order to maintain the orientation of the tool (parallel to itself). PAR 260 determines the range in which the compensation algorith is executed: (± 90° – PAR 260) < *pitch angle* < (± 90°+ PAR 260). Average range: 0–20.
260+*axis*	These parameters define the **Maximum Position Error** per axis, in encoder counts, which is allowed for the completion of MOVED, MOVELD, MOVECD, MOVESD and SPLINED commands. (These parameters are active only in the EXACT mode.) If *axis* is a servo gripper, this parameter defines the maximum fluctuation of the encoder value while the gripper is blocked.

Parameter Table 2	
Parameter	**Description**
274	**Pneumatic Gripper (OPEN/CLOSE) Configuration** Defines the controller output to which the pneumatic gripper is connected. Thus PAR 274 may have a value of 1–16. Allows the use of OPEN and CLOSE commands from keyboard and from teach pendant for controlling pneumatic gripper. If gripper opens in response to a CLOSE command (and vice versa, due to incorrect I/O wiring connections), the sign of the parameter value should be reversed; for example, PAR 274=–1. If PAR 274=0: pneumatic gripper not installed; not defined.
275	**Gripper Holding Power**. A constant analog output value to be applied to the gripper after the completion of a close gripper command. Be careful not to set this value too high. Typical value: 1000 or lower. Valid only for a servo gripper *without* encoder feedback.
276	**DAC Value** to be applied **at the start of gripper movement** for the amount of time specified in PAR 277.
277	**Duration of PAR 276**. Time value, defined in hundreths of a second.
280+*axis*	**Manual Movement Torque Limit**. The maximum torque allowed while moving the axes manually (by means of TP or keyboard). DAC value: 0–5000.
294	**Manual Cartesian Movement** maximum speed. A percentage of maximum linear speed, as defined by PAR 536.
300	**Keyboard Stroke Rate**. When operating robot in Manual keyboard mode, time required to hold down key before character is repeated, thus producing a smooth, continuous axis movement.
301	X coordinate of the rotation axis of arm link 2 when the robot is in the home position. Defined in microns.
302	Y coordinate (offset from center along the Y-axis) of the TCP when the robot is in the home position. Defined in microns.
303	Z coordinate of the rotation axis of arm link 2. Defined in microns.
304	Length of arm link 2 (from the first articulated joint). Defined in microns.
305	Length of arm link 3 (from the second articulated joint). Defined in microns.
306	Distance from pitch axis to the TCP (for **ER IX, MK2**); or, Distance from flange to the TCP (for **ER 14**). Defined in microns. Calcuated from PAR 307 and PAR 308.
307	Distance from pitch axis to flange (for **ER IX**, **MK2**); defined in microns. For **SCORA-ER 14**, PAR 307 is set to 0. The value of this parameter must be set to the exact dimension of the specific robot. The TOOL command then sets parameters 308, 309, 310 and 306 to the proper values.
308	Length value from TOOL command. Distance from flange to the TCP; defined in microns.

Parameter Table 2	
Parameter	**Description**
309	Offset value from TOOL command. Distance from the axis of symmetry of the flange to the TCP; defined in microns.
310	Angle value from TOOL command. Angle of TCP relative to the horizontal position when link 4 is horizontal (**ER IX** and **MK2**) and roll is 0 (all robots); defined in thousandths of a degree.
320+*axis*	**Driver Hardware Gain**. Gain of the PWM preamplifier of the driver card. Range: 0–15. Typical value: 1 or 2.
340 *Reserved*	*Reserved for debugging purposes. Do not alter values.* **Shift of Position Error for Adaptive Gain Tachometer**. Position error is shifted according to this parameter and the adaptive value of the tachometer gain is calculated taking into account the shifted value of position error. This parameter is used when the robot is not executing a movement.
340+*axis*	**Tachometer Hardware Gain**. Selects one of 16 values of gain for tachometer feedback. If PAR 340+*axis* is in the range 0–15: The value is downloaded directly to the hardware gain of the tachometer. If PAR 340+*axis* > 15: A more complex calculation of tachometer gain is calculated according the desired velocity and the position error at every moment. (Adaptive Tachometer Gain)
360 *Reserved*	*Reserved for debugging purposes. Do not alter values.* **Shift of Position Error for Adaptive Gain Tachometer**. The position error is shifted according to that parameter and the adaptive value of the tachometer gain is calculated taking into error the shifted value of position error. This parameter is used while the robot is executing a movement.



ACL Controller-B Control Loop

Parameter Table 2	
Parameter	**Description**
360+*axis*	**Integral Limit**. Highest value of analog output allowed for integral feedback. If the analog output value is greater than this value, the integral feedback is not incremented. Range: 0–5000.
380+*axis*	**Feed Forward**. A constant which represents the average value of the ratio: *analog output ÷ encoder counts*. Range: 0–20,000.
400+*axis*	Sets the **Speed of Search** for the index pulse. If par=0: An index pulse does not exist for the specified axis.
420+*axis*	**Index Pulse Position**. Records the distance, in encoder counts, between the home switch transition and the index pulse position. During initial operation this value is 0. After the first homing of each axis, this parameter automatically records the detected value. This value is then used by future home operations for verification. If the motor encoder or any other mechanical component is changed, set this parameter to 0 for the specific axis and rehome. This will insert the new value automatically into the proper parameter.
440+*axis*	The number of encoder counts for a **90° turn of motor**.
460+*axis*	**Speed of Search** for the home switch. The sign of this parameter also determines which side of the home switch is sensed as the homing position. If PAR 460+*axis*=0: search for home switch is not allowed for the specific axis.
500+*axis*	**Manual Movement Deceleration**. The rate of deceleration after the manual movement key is released. Defined as a percentage of maximum acceleration, as defined by PAR 520.
520+*axis*	**Maximum Acceleration/Deceleration** allowed for each axis during movement. In units of *encoder counts / (clock tick)*^{2}.
533	**Maximum Linear Acceleration** in microns/(hundreths of a second)^2.
534	**Maximum Pitch Acceleration** for **ER IX, MK2**, in millidegrees/(hundreths of a second)^2. **Maximum Roll Acceleration** for **ER 14**, in millidegrees/(hundreths of a second)^2.
535	**Maximum Roll Acceleration** for **ER IX, MK2**, in millidegrees/(hundreths of a second)^2. Not used for **ER 14**.
536	**Maximum Linear Speed** in (microns/second).
537	**Maximum Pitch Speed** for **ER IX, MK2**, in (millidegrees/second). **Maximum Roll Speed** for **ER 14**, in (millidegrees/second).
538	**Maximum Roll Speed** for **ER IX, MK2**, in (millidegrees/second). Not used for **ER 14**.
540+*axis*	**Maximum Encoder Range**. An envelope value used for various calculations. This value should be set to more than twice the maximum range of axis motion.
560+*axis*	**Home Switch Polarity.** Defines which values are used to indicate the home switch status: If PAR 560+*axis*=0, when the home switch is activated: `HS[`*axis*`]`=1 If PAR 560+*axis*=1, when the home switch is activated: `HS[`*axis*`]`=0

Parameter Table 2	
Parameter	**Description**
580+*axis*	**Deceleration Smoothness.** Used to adjust the smoothness of the stop following an abort (by A or CLRBUF command). In units of *encoder counts / (clock tick)*^{2}. Typical value: 5 (smooth stop) – 100 (abrupt stop).
600+*axis*	**Analog output value** to be applied to the axis when performing a hard home search. If PAR 600+*axis*=0: Hard home not allowed for specified axis.
640+*axis* *Reserved*	*Reserved for debugging purposes. Do not alter values.* **Tachometer Gain Maximum Value**. In cases where an adaptive tachometer is used (PAR 340+*axis*>15), this parameter defines the maximum value of the tachometer gain allowed for that axis.
660+*axis*	**Maximum DAC Value**: The absolute value of analog output that can be applied to the axis. Used to prevent the speed of the axis from exceeding a safe limit.
680+*axis*	**Servo Error.** At run time and during homing, if a position error is greater than that defined by the parameter, an impact condition is detected. Typical values for **ER IX** / **ER 14** / **MK2** axes: 1000–2000 encoder counts.
700+*axis* 720+*axis* 740+*axis* 760+*axis*	**Manual Movement Torque Limitation.** Whenever the axis encoder moves more counts than the value of PAR 700+*axis*, the torque value is sampled, and written to the system variable T0. In addition, the actual torque, *T*, is measured at each clock tick, and compared to the stored value T0. Each time (T – T0) > T_{max}, an internal counter is incremented. When the value of the counter equals the value of PAR 740+*axis*, an impact condition is detected. Maximum allowed torque is defined as: T_{max}=(PAR 720+*axis*) + [(PAR 760+*axis*) x *acceleration*]
780+*axis*	**Servo Error.** During manual movement, if a position error is greater than that defined by the parameter, an impact condition is detected. Typical values for **ER IX** / **ER 14** / **MK2** axes: 1000–2000 encoder counts.