ACL-Win

for Controller-BRC

User Manual

Catalog # 100354 Rev. A

intelitek

Copyright 2001 Intelitek Inc.

Catalog # 100354 Rev. A

December 2001

(Software v.1.2.38)
(Manual v.06 - May 2000)

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

Intelitek bears no responsibility for errors that may appear in this publication and retains the right to make changes to the software, hardware and manual without prior notice.

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

http://www.intelitek.com
info@intelitek.com

Table of Contents

1

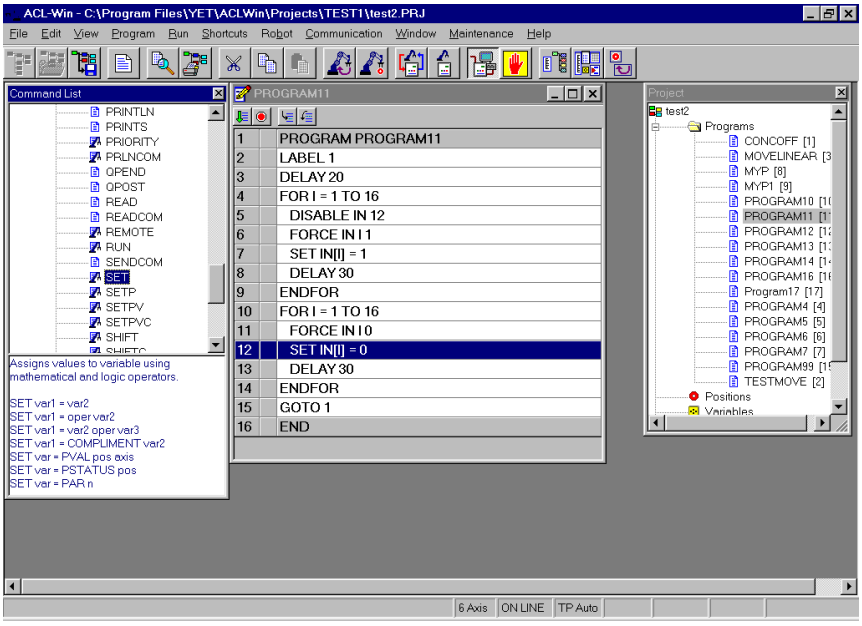
Getting Started

Introduction

ACL, Advanced Control Language, is an advanced, multi-tasking robotic programming language, which is programmed onto a set of EPROMs within the robot controller.

ACL-Win is the software interface which provides access to the controller from a PC, and provides all the functions needed to configure, program and operate the robotic system.

The teach pendant is a hand-held terminal which is also used for controlling the robot and peripheral equipment connected to the controller. The teach pendant is most practical for moving the axes, recording positions, sending the axes to recorded positions and activating programs. Other functions can also be executed from the teach pendant. You can easily alternate between the teach pendant and ACL-Win while working with the robotic system. The user manual supplied with the teach pendant fully describes the elements and functions of the teach pendant.



Installation

System Requirements

For best performance, the following system is recommended:

- An IBM-compatible PC with Pentium 200 MHz, or faster, processor.
- At least 64 MB of RAM.
- A hard drive with at least 50 MB of free disk space.
- Windows 95 Service Pack 2, Windows 98, or Windows NT.
- Super VGA (800x600) or better graphics display, minimum 256 colors.
- A mouse or other pointing device.

Installing the Software

The ACL-Win software is supplied on a CD-ROM disk.

To install the software on a PC which does not have a CD drive, do the following:

- Access a PC with a CD drive which is networked with the target PC, and follow the installation instructions in the section above.

OR

- Use a PC which has both a CD drive and a floppy drive.
    1. Copy all files in the CD's **Install** folder onto diskettes:
    2. Each file named **data*n*.cab** (where *n* is a number; i.e., data1, data2) will fill one diskette.
    3. All other files can be copied together onto a single diskette.
    4. Copy all files from all the diskettes into a temporary folder on the target PC.
    5. From the temporary folder, run **setup.exe**.
    6. Follow the instructions that appear on the screen.

    You can delete the temporary directory after the installation, or keep it for reinstalling the software

To install the software from the CD-ROM disk, do the following:

1. Start Windows.
2. Close any applications that are open before you begin the installation.

3. Insert the CD-ROM into its drive. The installation should start automatically.

   If the install does not start automatically, select the **DISK1** subdirectory and run **SETUP.EXE**.

4. Follow the instructions that appear on the screen.

During the software installation messages and a percentage bar will be displayed on the screen to reflect the status of the installation procedure.

By default, ACL-Win is installed in Program Files/Yet/ACLWin. When the installation is complete, the ACL-Win program group is displayed on the Windows desktop.

Uninstalling the Software

To uninstall the ACL-Win software, do either of the following:

- Click the Uninstall icon in the ACLWin program group and follow the instructions and prompts on your screen..

OR

- Use the Windows Add/Remove Programs feature:

  1. Select **Start | Settings | Control Panel | Add/Remove Software**.
  2. In the Add/Remove Programs dialog box, select ACL-Win and then click Add/Remove.
  3. Follow the instructions, and respond to the prompts on your screen.

Activating the Software

ACL-Win can be operated offline or online.

In **offline operation** ACL-Win and controller do not communicate, even if the PC and controller are connected. Offline operation allows you to program the robotic system without connecting the PC to the robot controller. Offline programming allows the robotic system to continue operating while new programs are being prepared.

In **online operation**, the PC communicates actively with the controller through ACL-Win. When ACL-Win is online, the data in both the PC and the controller must be identical to ensure proper operation.

Data can be altered in the controller while it is operating as a stand-alone unit. Data can also be manipulated in the PC when the software is operating offline. To ensure that data is identical when online operation begins, the system uploads data from the controller to the PC, or downloads data from the PC to the controller.

Starting Offline Operation

If you intend to operate ACL-Win offline (without communicating with controller), simply click the ACL-Win icon to activate the software.

ACL-Win loads in offline mode, even when the controller is connected and turned on.

Starting Online Operation

If you intend to operate ACL-Win online (communicating with controller), do the following:

1. Make sure the controller's RS232 cable is connected to one of the PC's COM ports.
2. Turn on the controller and the PC.
3. Click the ACL-Win icon to activate the software.
4. At the prompt, select English or Japanese for the interface language.
5. Make sure settings in the **Communication | Settings** menu are correct.
6. Select **File | Open Project.**
   - Select a project file.
   - Select **Communication | Online**.
   - In the Online Transition dialog box, do either of the following:
     - Select **Download** for all data options to transfer data from PC to controller.
     - Select **Upload** for all data options to transfer data from controller to PC.

   OR

   Select **File | New Project**.
   - Select **Communication | Online**.

     When no project data exists in PC memory when the system goes online, all data in the controller is *automatically* uploaded to the PC during the transition to online mode. No prompt appears.

For more complete instructions on online and offline operation, and download and upload functions, see Chapter 8, "System Configuration."

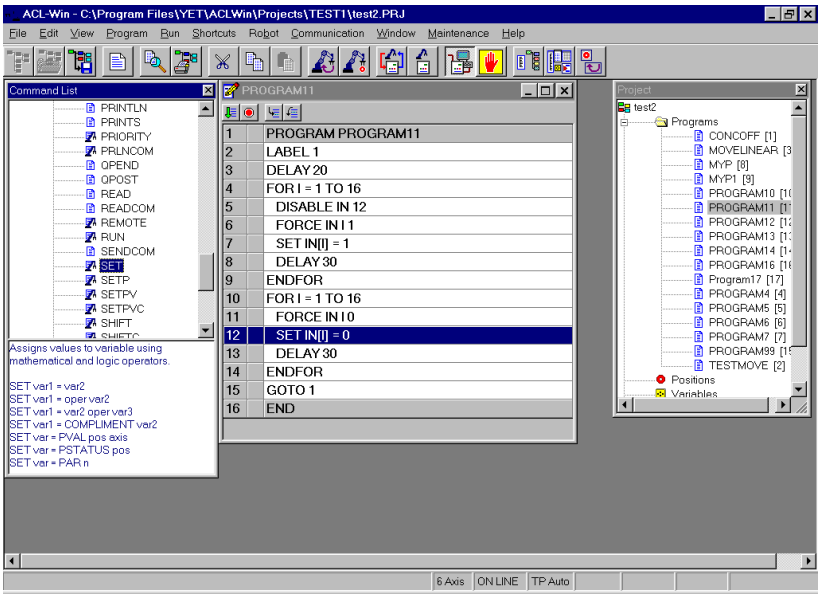
2

ACL-Win Overview

ACL Projects

An ACL-Win **project** comprises nearly all the contents of the robot controller. It includes programs, positions, variables and configuration data.

ACL-Win always has a currently active project. The software always opens with a new, empty project. The user may open a project from PC files or upload it from the controller. Only one project can be open at a time.

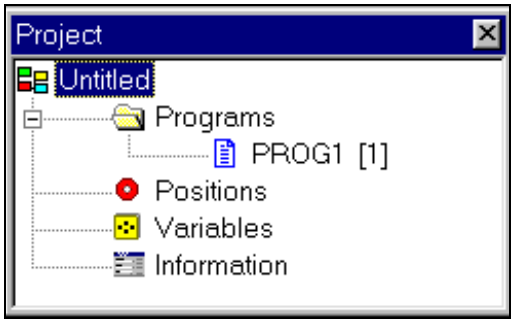


When ACL-Win is opened, the following windows are displayed:

- Project Tree
- Command List
- Program window

Project Tree

The **project tree** provides an overview and access to the components of an ACL-Win project.



Programs	Double-clicking expands and contracts the Programs folder. Programs are listed in the order of their program ID numbers. This number is automatically assigned by the system and may be used for accessing programs from the teach pendant.
Prog_name	Opens a Program window and displays the ACL code of the specified program.
Positions	Opens the Positions List window for displaying and manipulating positions.
Variables	Opens the Variables window for displaying and manipulating defined variables.
Information	Opens the Project Information dialog box for entering descriptive information about a project.

The various types of project data – programs, positions and variables – are managed and manipulated through dialog boxes.

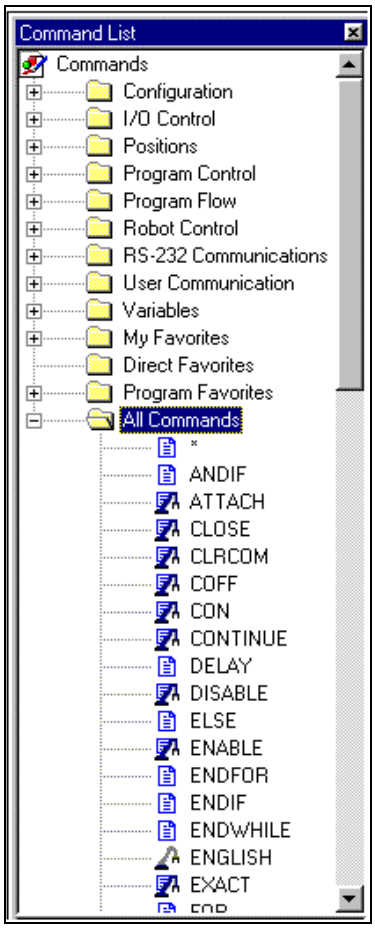
Command List

The **command list** provides access to all ACL commands.

The command list contains a number of folders. Click on a folder to expand

and contract it.

- The first set of folders are fixed groups of commands arranged according to function.
- The second set of folders are available for the user to form groups of “favorite” commands. Select and right click on a command to access the options for manipulating the command. Commands deleted from a favorite folder remain available in the other folders.
- The last folder is a complete listing of all ACL commands, arranged alphabetically.



Each command is marked by one of three icons, which indicate whether the command may be used in Program (edit) mode or Direct mode, or both:

Program (edit) command. The command can only be inserted into the program currently being edited.

Direct command. The command can only be sent to the robot controller for immediate execution.

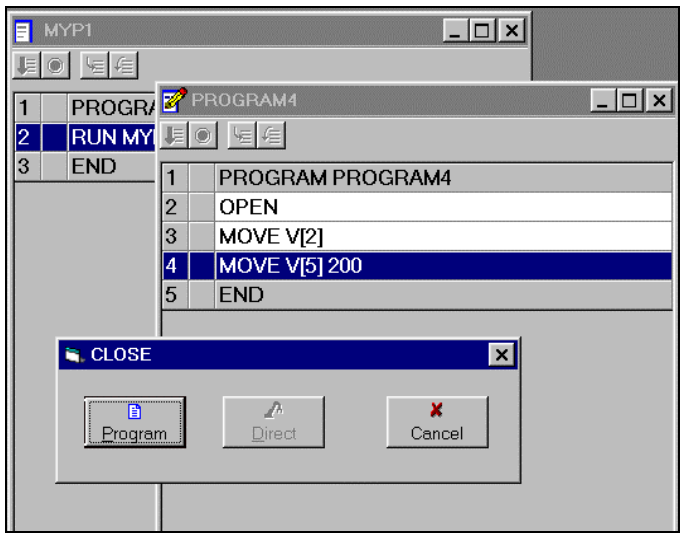
The command can be written to a program or directly executed.

Double-clicking on a command usually opens a dialog box, in which you must specify required and/or optional parameters for the particular command.

Dialog boxes include Direct and/or Program buttons to allow you to send the command to the appropriate destination. These buttons remain disabled until you have entered all required parameter fields. The Direct button remains disabled when the software is operating offline.

Program Window

Each Program window displays one ACL program. Many program windows can be opened simultaneously, although only one can be active. The currently active program is indicated by the “writing pencil” icon in the window’s title bar. When a command dialog box is opened, the “pencil” icon continues to indicate the active program, in which the command will be entered, as shown in the example below.



The main ACL-Win toolbar provides shortcut buttons for program editing functions – copy, paste and find/replace.

The Program window's toolbar has only a few functions. Right-clicking on the Program window opens its short-cut menu, which provides access to many more functions.

For more information on program editing and execution, see Chapter 4, "Programs."

ACL Menus

Application Toolbar

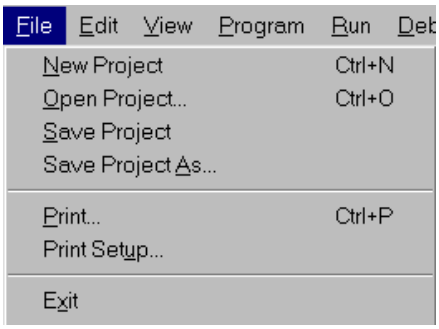
The main toolbar in the ACL-Win application window provides shortcuts to certain commands and functions.

	New Project	Creates a new project. Displays the Project Tree, the Command List and an empty Program window.
	Open Project	Opens an existing project from file. Displays the Project Tree, the Command List and all Program windows.
	Save Project	Saves some or all project data. Opens the Data Selection dialog box which allows you to choose the data to be saved.
	New Program	Opens an empty Program window.
	Find	Performs a search in the active Program window.
	Print Project	Prints project data.
	Cut	Cuts the selected code in a Program window and places it on the clipboard.
	Copy	Copies the selected code in a Program window to the clipboard.
	Paste	Inserts the clipboard contents at the selected point.
	Move	Opens the Move dialog box.
	Teach Position	Opens the Teach Position dialog box.

	Control On	Executes ACL command CON; servo control on.
	Control Off	Executes ACL command COFF; servo control off.
	Online / Offline	Toggle button. Switches between online and offline controller operation.
	Abort	Immediately aborts execution of all running commands and programs.
	Edit Layout	Displays a set of dialog boxes and windows used for editing programs, as defined in the EDIT.LAY file.
	Debug Layout	Displays a set of dialog boxes and windows used for debugging programs, as defined in the DEBUG.LAY file.
	Teach Layout	Displays a set of dialog boxes and windows used for teaching positions and moving the robot as defined in the TEACH.LAY file.

File Menu

The File menu contains the standard Windows functions and shortcuts for managing and printing project and data files.



Each project should reside in its own directory. Two different projects may reside in the same directory if they want to use the same data files.

The project file name is user-defined, but always has the extension **prj**.

Project data files (variables and positions) automatically receive the same name as the project and a predefined extension, as follows:

- **pos** for position files
- **var** for variable files

The names of these data files may not be changed. Project data files are always saved to the project directory.

ACL programs are named sequentially (Program1, Program2, Program3, and so on) and receive the extension **prg**. ACL program files may reside anywhere, and may be specified using either a full or a relative path name. This allows common programs to be shared by different projects.

ACL program files are managed through the **Program** menu (and not the File menu).

When loading data from the disk to the current project, you will be prompted to confirm a message that the operation will overwrite any data that has been changed and not yet saved.

New Project

Creates a new, empty project. Displays the Edit Layout.

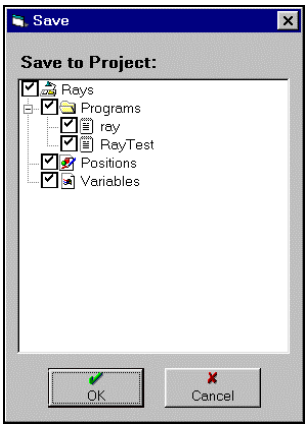
Open Project

Opens a dialog box for selecting the project to be loaded. Once a project is selected, displays the Edit Layout.

Save Project

Opens a dialog box for selecting the data to be saved with the project file.

The Data Selection dialog box allows you to choose one or more types of data to be loaded, saved, uploaded or downloaded. Data can be transferred individually or in a batch. Individual programs can also be selected.



To save or load data files to or from the PC memory, use the File menu options (Save/Open).

Save Project As

Opens a dialog box for defining a new or different file name for the project.

Print

Opens a Data Selection box for selecting the project data to be printed.

Print Setup

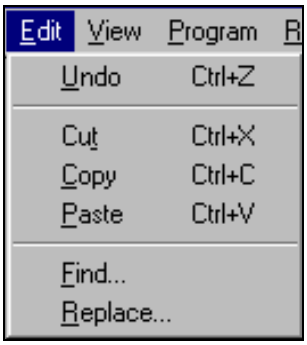
Opens standard Windows printer setup dialog box.

Exit

Closes ACL-Win.

Edit Menu

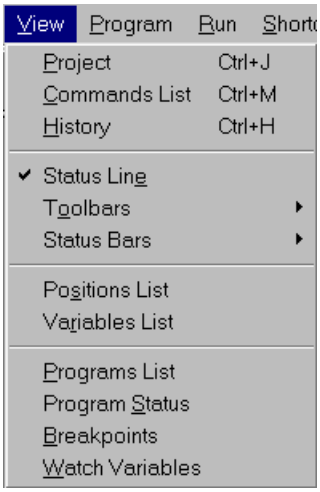
The Edit menu contains the standard Windows functions and shortcuts for editing program lines.



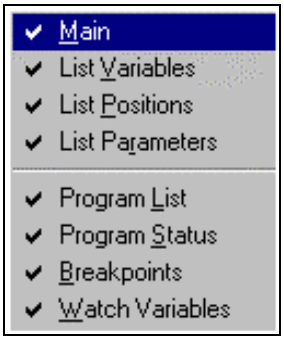
	Undo	Reverses the last action or operation, such as adding, deleting or modifying a program line. Up to 20 operations can be undone.
	Cut	Cuts the selected code in a Program window and places it on the clipboard.
	Copy	Copies the selected code in a Program window and puts it on the clipboard.
	Paste	Inserts the contents of the clipboard at the current point in the program.
	Find	Performs a search in the active Program window.
	Replace	Finds the specified string in the active Program window and changes it.

View Menu

The View menu allows you to display windows, dialog boxes and status bars. It also enables you to display or hide shortcut toolbars.

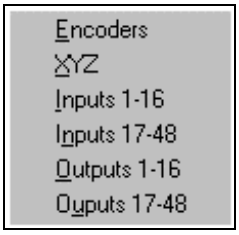


Project	Opens the Project Tree..
Command List	Opens the Command List.
History	Opens the History window. The History window displays a list of the ACL commands most recently entered by the user, and indicates whether the command was entered in Program or Direct mode. Commands can be selected from this list by double-clicking on them.
Status Line	Displays the system status line.
Toolbars	Displays toolbars. You must select the toolbar display separately for each window or dialog box.



The bottom four options are not available when the system is operating offline.

Status Bars	Displays status bars for encoders, XYZ coordinates, inputs 1-16, inputs 17-48, outputs 1-16, outputs 17-48. You must select the display separately for each status bar.

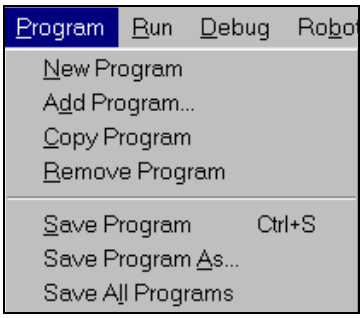


These options are not available when system is operating offline.

Positions	Opens the Positions List dialog box.
Variables	Opens the Variables List dialog box.
Parameters	Opens the Parameters List dialog box.
Program List	Opens the Program List dialog box.
Program Status	Opens the Program Status dialog box.
Breakpoints	Opens the Breakpoints dialog box.
Watch Variables	Opens the Watch Variables dialog box.

Program Menu

The Program menu allows you to manage ACL program files.



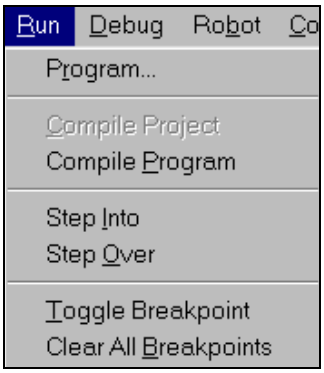
New Program	Opens a new, empty Program window and adds it to the current project.
Add Program	Opens a dialog box for selecting an existing program file and adds it to the project.
Copy Program	Opens the Save As dialog box. Makes a copy of the currently active program in the current project.

Remove Program	Removes the program currently selected in the Project window from the current project.
Save Program	Saves the currently active program.
Save Program As	Saves the currently active program under a new or different file name.
Save All Programs	Saves all programs listed in the project tree.

For more information on ACL programming, see Chapter 4, “Programs.”

Run Menu

The Run menu allows you to execute and test ACL programs.



Program	Opens the RUN command dialog box for selecting a program to be run.
Compile Project	Validates the project as a whole, including all programs and inter-program rules. Not available when controller is online.
Compile Program	Validates the code in the currently active program window.
Step Into	Single steps program execution (executes the current command line and stops). When used at a GOSUB command, Step Into opens the called subroutine’s Program window, and stops on the subroutine’s first line.
Step Over	Single steps program execution (executes current command line and stops). When used at a GOSUB command, Step Over calls and executes the subroutine in its entirety, and then stops on the command line following the GOSUB command.

Toggle Breakpoint	Inserts or deletes a breakpoint at the selected line. Breakpoints are used to stop execution of a program at a specific point and allow single-step execution of command lines which follow the breakpoint
Clear All Breakpoints	Deletes all currently defined breakpoints, in all programs in the active project.

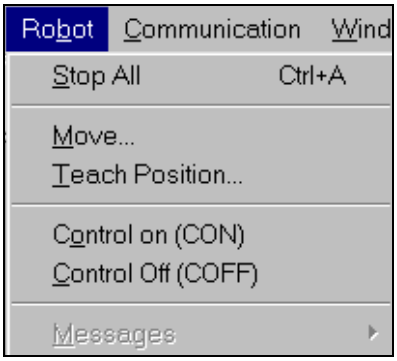
For more information on program execution and debugging, see Chapter 4, “Programs.”

Shortcuts Menu

The Shortcuts menu provides **keyboard** access to all the popup (right-click) menus in ACL-Win.

Robot Menu

The Robot menu provides access to some of the most commonly used robot operations.



	Stop All Ctrl+A	Aborts all running programs and robot movements
	Move	Opens the MOVE command dialog box.
	Teach Position	Opens the Teach Position dialog box.
	Control On	Executes ACL command CON; servo control on.
	Control Off	Executes ACL command COFF; servo control off.

Communication Menu

The communication menu is used to define settings and establish communication between the controller and the PC and to transfer data to and from the controller.

Online	Toggles online and offline controller operation.
Settings	Displays the Communications Settings dialog box, which allows you to define the RS232 serial communication ports to enable PC–controller communication. **Do not change any other setting!**

For more information on online and offline operating, and uploading and downloading, see Chapter 8, "System Configuration."

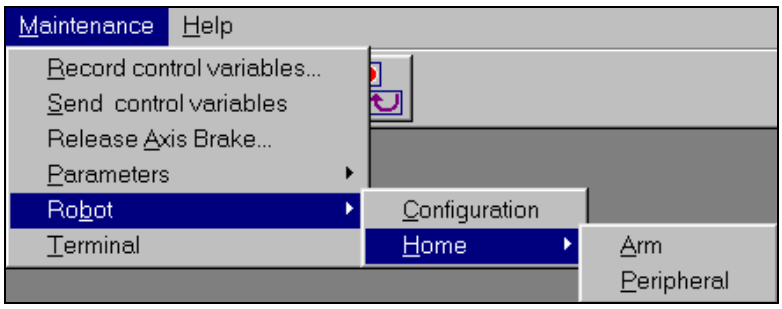
Window Menu

The Window menu contains the standard Windows options for displaying or activating open windows.

Cascade Tile Horizontal Tile Vertical Arrange Icons 1-5 (and More)	Standard Windows commands for all open windows.
Close All	Closes all open windows.
Save Layout	Opens a dialog box for saving the current screen layout to a file. Saves the size and position of all windows. Also saves column grid widths. Screen layout settings can be saved to the EDIT, DEBUG, TEACH layout files, or to a user-defined file.
 Load Layout	Opens a dialog box for selecting and loading a saved screen layout (.**lay**) file.

Maintenance Menu

The Maintenance menu provides direct access to controller functions that are not part of ACL projects.



Record Control Variables	TBC
Send Control Variables	TBC
Release Axis Brake	TBC
Parameters	Opens the Parameters List dialog box.
Robot	**Configuration**: Opens the Robot Controller Configuration dialog box. **Home**: Sets the current location of the robot or peripheral axis as the Home reference position.
Terminal	Opens an ACL-DOS window.

Status Indicators

System Status Line

By default, the system status bar is always displayed. The system status bar can be toggled on and off by selecting View | Status Bar.

The status bar has six fields:

Messages	Context-specific system messages (eg, successful completion of Direct Mode command).
Configuration	Shows the number of robot axes and the name of a peripheral device, if configured.

On-Line Status	Shows the control status of the robot axes: • Off Line • On Line
TP Status	Shows the status of the teach pendant: • TP Teach: Teach pendant is switched to Teach, and has control. • TP Auto: Teach pendant is switched to Auto; ACL-Win software has control.
Emergency	Indicates whether or not the system is in Emergency state. • EMERG: Emergency state. • (blank): no Emergency.
Global Speed	Indicates the current global speed, in percentages.
Joint Speed	Indicates the current joint speed, in percentages.
Linear Speed	Indicates the current linear speed, in mm/sec.

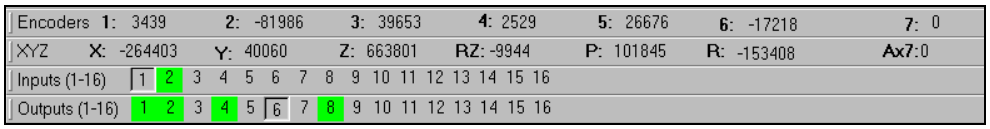
Status Bars

Status bars can be displayed at the bottom of the application window to show position and I/O data.

Status bars are not available in offline operation.

You can rearrange the order of the status bars, or place one on top of the other, by clicking on the status bar handle or title and dragging the bar to the desired location.

The refresh rate for the data in the status bars is defined in the ACL-Win.ini file. The default refresh rate is once every 4 seconds. However, whenever a controller event occurs, the display is updated immediately.



Encoders Status Bar

The Encoders status bar displays the current encoder values of each of the robot's axes and the peripheral axis.

The display of the encoders status bar can be toggled on and off by selecting View | Status Bars | Encoders.

XYZ Status Bar

The XYZs status bar displays the current position of the robot's TCP in World coordinates and the encoder value of the peripheral axis.

The display of the encoders status bar can be toggled on an off by selecting View | Status Bars | XYZ.

Input / Output Status Bars

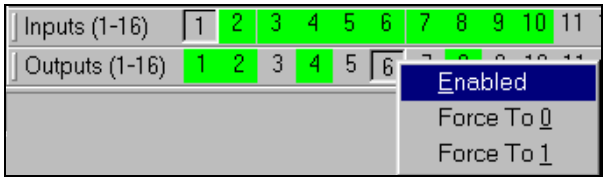
The Input/Output Status Bars display the status of each of the digital inputs and outputs. There is one status line for each of the following sets of data:

- Inputs 1-16
- Outputs 1-16
- Inputs 17-48 (if installed)
- Outputs 17-48 (if installed)

Input/Output status is indicated as follows:

- Green indicates ON.
- Gray indicates OFF.
- Depressed (sunk) indicates DISABLED

Right-click on a specific I/O number on the status bar to enable/disable the specific I/O pin.



If an I/O is disabled, it can then be forced to either 0 (off) or 1 (on).

The display of the I/O status bars can be turned on and off by selecting View | Status Bars.

3

Positions and Movements

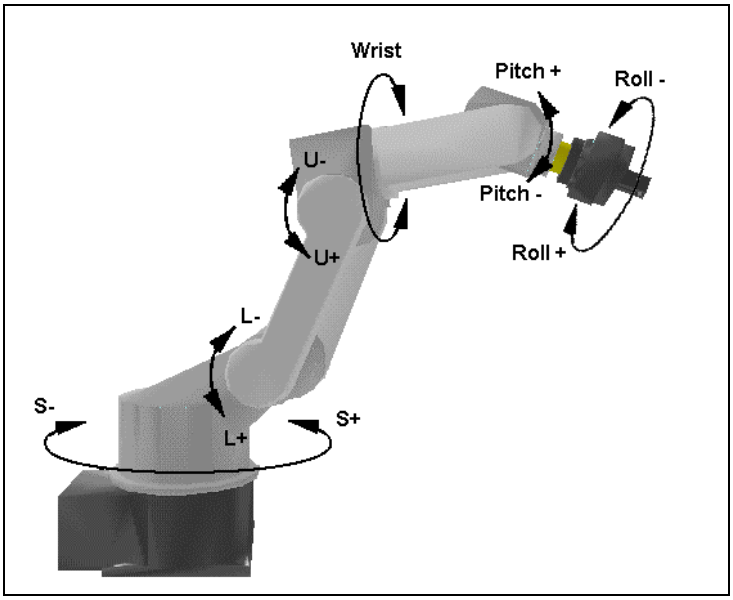
This chapter describes the different coordinate systems used for recording positions and moving the robotic axes.

Coordinate Systems

ACL allows robotic systems to be operated and programmed in three different coordinate systems: **Joint** coordinates, **World** (XYZ) coordinates and **Tool** coordinates.

The six robot axes (joints) are defined as shown in the figure below.

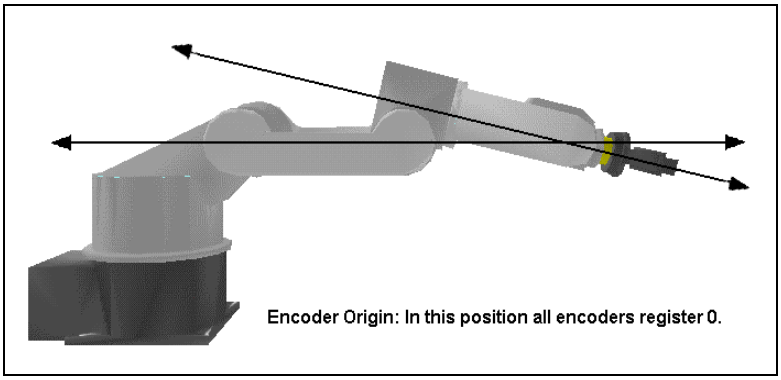
1. Base S- / S+
2. Lower arm L- / L+
3. Upper arm U- / U+
4. Z-Roll (wrist)
5. Pitch
6. Roll



Joint Coordinate System

Joint coordinates specify the location of each axis in encoder counts.

All encoders register 0 when the robot axes are in the “horizontal” position, as shown in the figure below.



Axis movement in the positive direction is upward relative to the “horizontal” position or counterclockwise as viewed from above.

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

World (XYZ) Coordinate System

World coordinates are relative to the robot’s point of origin.

World (XYZ) coordinates specify the location of the robot's tool center point (TCP). The TCP is defined by its distance from the robot’s point of origin (the center bottom of the robot base) along three linear axes (X,Y and Z) and by the three rotational axes (RollZ, Pitch and Roll).

Tool Coordinate Systems

Tool coordinates allow movement of the robot in directions associated with the tool coupled to the robot flange.

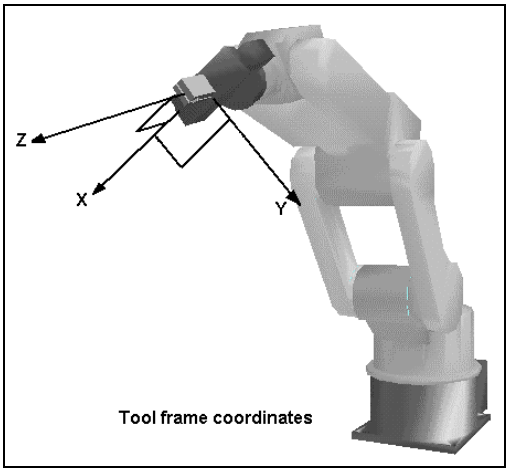
The tool coordinate system is a frame fixed to the robot flange; that is, the frame’s orientation changes whenever the flange orientation changes.

The orientation of the tool frame is relative to the robot flange, and is defined by default controller parameters, as follows:

With the robot standing at the home position ($HOME):

- Tool Z-axis is perpendicular to the flange surface.
- Tool X-axis is along the world Y axis, in the negative direction.
- Tool Y-axis is along the world Z axis, in the negative direction.

The orientation of the tool frame can also be defined by the user by means of the ZTOOL command:



Positions

Positions are reserved memory locations which hold position data.

Types of Positions

ACL recognizes and uses eight types of positions:

- **Absolute Joint**

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

- **Absolute XYZ**

  Position data are the coordinates of the position in World (XYZ) values.

- **Relative to Another Position by Joint**

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

- **Relative to Another Position by XYZ**

  Position data are the differences between the World (XYZ)coordinate values of one position and the World (XYZ) coordinate values of another position.

- **Relative to Another Position by Tool**

  Position data are the differences between the Tool coordinate values of one position and the Tool coordinate values of another position.

- **Relative to Current by Joint**

  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**

  Position data are calculated by adding the World (XYZ) coordinate values of one position to the World (XYZ) coordinate values of the current position.

  The current position is the World (XYZ) coordinate values at time the command using the position is executed.

- **Relative to Current by Tool**

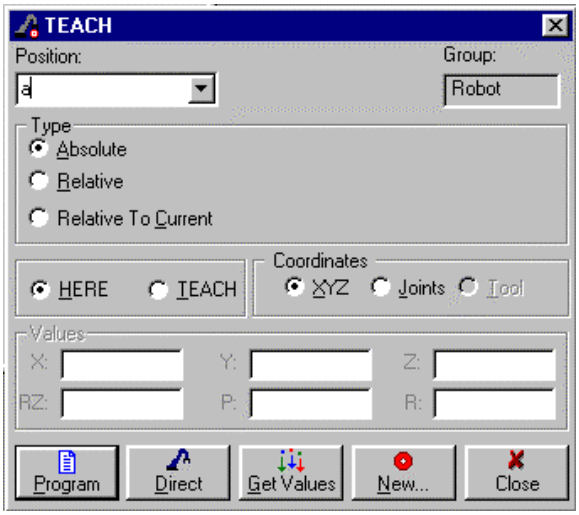
  Position data are calculated by adding the World (XYZ) coordinate values of one position to the World (XYZ) coordinate values of the current position.

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

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.

Recording Positions

The TEACH and HERE commands activate the same dialog box, which is used for recording positions.



To access this dialog box, do any of the following:

- From the command list, select the command HERE or TEACH.
- Select **Robot | Teach Position**.
- Click the Teach Position button on the main menu bar.

The HERE/TEACH/Teach Position dialog box has the following functions and options:

Position	A list of all user-defined and system-defined positions. Select the position whose coordinates you want to record. If you need to define (create) a position which does not appear on the list, select New.
New	Opens the Define Position dialog box for naming and defining a new position.

Type	Select the type of position you want to record. • **Absolute**. • **Relative to**. Opens another list of all defined positions. Select the position which be the reference point for the relative position. • **Relative to Current**.
Group (Axes)	Positions are defined for a particular axis control group – either the **robot** axes or a **peripheral** axis. When a peripheral axis is not installed or configured, the options are not available.
HERE	**HERE** (and its variants) causes the controller to record position coordinates according to the current location of the robot or peripheral axes. HERE can be used as either a Direct or a Program command.
TEACH	**TEACH** (and its variants) requires the user to enter values for position coordinates. TEACH can be used only as a Direct command
Coordinates	Select the type of coordinates you want to record. • **XYZ** (World) coordinates. • **Joints** (encoder units). • **Tool** coordinates.
Values	Displays the coordinates of the current or selected position. Depending on the type of position coordinates you have selected, Values are for either joints **1**, **2**, **3**, **4**, **5** and **6** or world axes **X**, **Y**, **Z**, **RZ**, **P** and **R**.
Get Values	Gets the coordinates of the axes' current location from the robot controller. Displays the values acccording to the type of coordinates selected.
Program	Available when all required options and fields are selected and completed.
Direct	Available when all required options and fields are selected and completed.
Close	Closes the TEACH/HERE dialog box. Unlike other command dialog boxes, this dialog box remains open after you select Program or Direct. This allows you to record a series of positions without having to reopen the dialog box repeatedly.

By selecting various combinations of options in the TEACH/HERE dialog box, you will execute the commands shown in the table below.

To Record this TYPE of Position….	**When …**	**Select COMMAND** / **Select COORDINATES**		**Select DIRECT to execute.** / **Select PROGRAM to write …**
Absolute Joints	CONTROLLER records current coordinates	HERE	Joints	HERE
Absolute XYZ		HERE	XYZ	HEREC
Absolute Joints	USER defines coordinates	TEACH	Joints	
Absolute XYZ		TEACH	XYZ	
Relative to Another Position by Joints	CONTROLLER records current coordinates	HERE	Joints	HERER
Relative to Another Position by XYZ		HERE	XYZ	HERERC
Relative to Another Position by Tool		HERE	Tool	HERERT
Relative to Another Position by Joints	USER defines coordinates	TEACH	Joint	
Relative to Another Position by XYZ		TEACH	XYZ	
Relative to Another Position by Tool		TEACH	Tool	
Relative to Current Position by Joints	USER defines coordinates	TEACH	Joints	
Relative to Current Position by XYZ		TEACH	XYZ	
Relative to Current Position by Joints		TEACH	Tool	

The commands **SETPV**, **SETPVC**, **SHIFT** and **SHIFTC** are used to change one coordinate of a previously recorded absolute position. Use the command list to access these commands; they cannot be entered through the Teach Position dialog box.

SETPV *pos axis var*

Changes the value of a recorded Joint position by one coordinate.

SETPVC *pos axis var*

Changes the value of a recorded XYZ or Tool position by one coordinate.

SHIFT *pos* BY *axis var*

Changes the value of a recorded Joint position by one offset value.

SHIFTC *pos* BY *axis var*

Changes the value of a recorded XYZ or Tool position by one offset value.

Although positions values are recorded in the Joint, World (XYZ) or Tool coordinate system, the axes can be instructed to move to positions in any coordinate system. The controller converts the coordinate values according to the movement command which is issued.

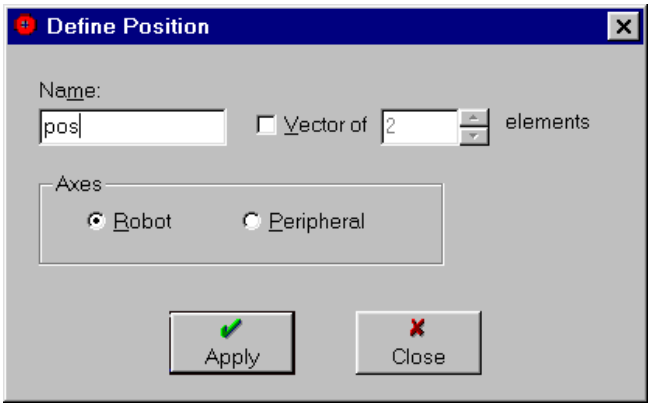
If a position is defined but not recorded, attempts to execute commands which refer to that position will cause run time errors.

Defining Positions

A position or position vector must be defined before coordinates can be recorded. To define a position is to reserve a location in controller memory and give a name to the location.

From the Teach Position dialog box, select **New**.

The Define Position dialog box opens.



.

ACL recognizes three types of position names:

- Alphanumeric names (such as P, POS10, A2). The name may be a combination of up to twelve characters, and must begin with a letter.
- Vector names (such as PVEC[50] and PVEC[10]) of up to twelve characters and an index. A position vector – an array of positions – can be attached to the teach pendant by means of the ACL command

ATTACH. The vector positions can then be accessed from the teach pendant by means of their index number.

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.

- Numerical names (such as 3, 22, 101) of up to twelve digits. These positions can also be defined from the teach pendant.

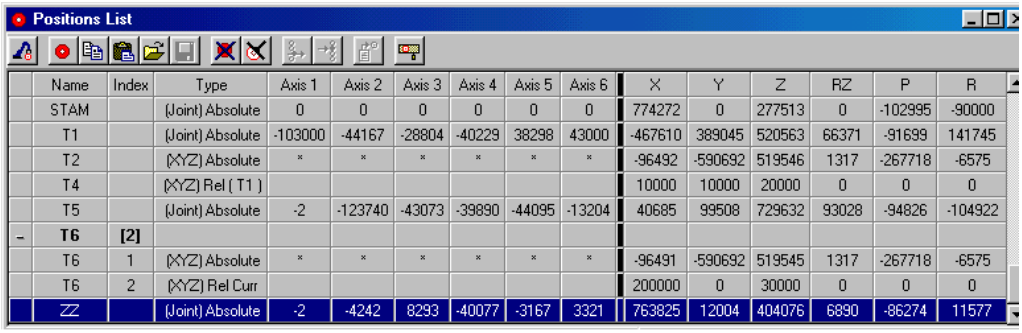
For more efficient programming, it is recommended that you define position vectors and record positions in the vector; that is, ***use vector indices rather than numerical names****.*

Once a position has been defined as a robot or peripheral position, it remains dedicated to that axis control group; the axis definition can be changed only by deleting and then redefining the position.

Apply	Creates new position according to name and definitions currently displayed in the dialog box..
Close	Closes the Define Position dialog box. Unlike most other dialog boxes, this dialog box remains open after you select Apply. This allows you to define a series of positions without having to reopen the dialog box repeatedly.

Listing Positions

The Positions List displays the position values and allows you to manipulate positions. It is automatically updated if the robot controller reports a change in position values.



	Name	Index	Type	Axis 1	Axis 2	Axis 3	Axis 4	Axis 5	Axis 6	X	Y	Z	RZ	P	R
	STAM		(Joint) Absolute	0	0	0	0	0	0	774272	0	277513	0	-102995	-90000
	T1		(Joint) Absolute	-103000	-44167	-28804	-40229	38298	43000	-467610	389045	520563	66371	-91699	141745
	T2		(XYZ) Absolute	*	*	*	*	*	*	-96492	-590692	519546	1317	-267718	-6575
	T4		(XYZ) Rel (T1)							10000	10000	20000	0	0	0
	T5		(Joint) Absolute	-2	-123740	-43073	-39890	-44095	-13204	40685	99508	729632	93028	-94826	-104922
-	**T6**	**[2]**													
	T6	1	(XYZ) Absolute	*	*	*	*	*	*	-96491	-590692	519545	1317	-267718	-6575
	T6	2	(XYZ) Rel Curr							200000	0	30000	0	0	0
	ZZ		(Joint) Absolute	-2	-4242	8293	-40077	-3167	3321	763825	12004	404076	6890	-86274	11577

To access this dialog box, select Positions in the Project tree, or select **View | Positions**.

The Position List short-cut menu provides the following functions:

Teach	Activates or opens the TEACH/HERE dialog box for recording a position.
New	Opens the Define Position dialog box for defining a new position.
Copy	Available in Off-line mode only. Copies a position from the current project. You are prompted to define a name and the number of vector elements for the new position. If a single coordinate (cell) is selected, only that specific coordinate will be copied.
Open	Available in Off-line mode only. Allows you to open a Position List from another project and to select a position to be copied into the current project. After selecting the position from the source project, click the Save button to copy the position into your current project.
Save	Available in Off-line mode only. Companion to Copy and Open functions. Selected from the Position List in the current or project. Saves the selected position into the current project. If a single coordinate (cell) is selected, only that specific coordinate will be saved.
Delete	Deletes the position currently selected in the Position List. If the selected position is a vector element, you will be prompted whether or not to delete the coordinates values of all the positions in the vector.
Clear Values	Clears the coordinate values of the position currently selected in the Position List. If the selected position is a vector element, you will be prompted whether or not to clear the coordinates values of all the positions in the vector.
Delete Vector Element and Compress	Deletes the vector position currently selected in the Position List, and renumbers the elements in the vector.

Insert Vector Element	Inserts a vector position at the line currently selected in the Position List, and renumbers the elements in the vector.
Reload from Controller	Uploads all positions and their coordinates from the controller to the PC. Overwrites current position coordinates in PC.
Peripheral Positions	Displays a list of all the peripheral positions which have been recorded. Select again to toggle the display back to Robot positions.

System Positions

System positions are predefined positions which are reserved for specific controller functions and calculations.

Position POSITION

POSITION is 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.

The following are examples of commands which access and utilize POSITION:

- SETP 100=POSITION

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

- 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 World (XYZ) coordinate value of the robot's current position.

You can also 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

- SETP POSITION=P[1000]

Position $HOME

This position contains the coordinates of the robot's current location when the **Robot | Home** command is executed.

Position $TCP[5]

This vector contains five positions which are used by the TCP command to calculate the robot's tool center point.

To record these positions, bring the robot's tool center point to the same location five times, with a different orientation each time. The TCP command will then determine the center of the sphere described by the five positions.

Positions $BOX_ORG, $BOX_X, $BOX_Y, $BOX_Z

These positions are used to define a parallelipedic workspace shared by two or more robots, used by the BOXOUT and BOXSTOP commands.

Positions $FRAME_ORG, $FRAME_X, $FRAMEX_Y

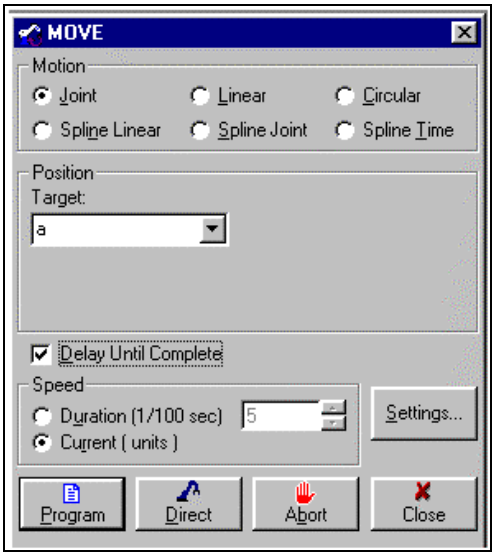
$FRAME_ORG and $FRAME_X are used to define the Tool X axis used by the TFRAME command.

FRAME_ORG, $FRAME_X and $FRAME_XY are used to define the Tool XY plane used by the TFRAME command.

Axis Movement

Programming Movement

The MOVE command activates a dialog box which allows you to write and execute commands for moving the robot and peripheral axis.



To access this dialog box, do any of the following:

- From the Command list, select the command MOVE.
- Select **Robot | Move**.
- Click the Move button on the main menu bar.

The dialog box has the following functions and options:

Motion

Select the type of movement.

- **Joint**
- **Linear**
- **Circular**
- **Spline Linear**
- **Spline Joint**
- **Spline Time**

The table which appears later in this section describes the differences between these options.

Position	A list of all user-defined and system-defined positions. Depending on the type of movement you select, additional fields appear, for selecting **Target, Start, End** and **Via** positions, and for specifying **Index** number(s) of vector positions.
Delay Until Complete	When selected, this option attaches the suffix **D** to all movement commands written to a **program.** When the (default) **Exact** mode is in effect, *move*D commands are completed only when the axes have arrived at the target position with the required accuracy, no matter how long it takes, and even when *duration* is specified. This ensures that operations defined in a program are executed sequentially.
Speed	Select the option which will determine the speed of movement: • **Duration (1/100 sec)**. Movement speed will be determined according to a time definition. Defined in hundredths of a second. • **Current**. Movement will be executed at current speed speeding.
Settings	Opens the Movement Settings dialog box which displays the current speed and accelereation settings.
Program	Adds command line to active program. Available when all required options and fields are selected and completed.
Direct	Executes command line in direct mode (online). Available when all required options and fields are selected and completed.
Close	Closes the MOVE dialog box. Unlike other command dialog boxes, this dialog box remains open after you select Program or Direct. This allows you to record a series of positions without having to reopen the dialog box repeatedly.
Abort	Immediate aborts movement.

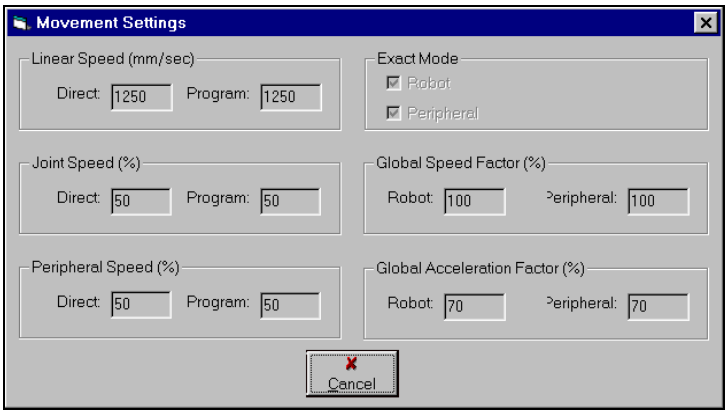
By selecting various combinations of options in the MOVE dialog box, you will write or execute the commands shown in the table below.

This table assumes the option **Delay Until Complete** is selected. (If this option is not selected, commands will be written without the D suffix.)

To produce a movement which ….	Select MOTION	Select SPEED	Select DIRECT to execute movement. OR Select PROGRAM to write …
Moves axes to target position at current speed.	Joint	Current	MOVE *pos*
Moves axes to target position within time specified.		Duration	MOVE *pos time*
Moves TCP to target position along a linear path at current speed.	Linear	Current	MOVEL *pos*
Moves TCP to target position along a linear path within time specified.		Duration	MOVEL *pos time*
Moves TCP through one position to target position along a circular path at current speed.	Circular	Current	MOVEC *pos pos*
Moves TCP through *or near* all consecutive vector positions within a specified range. Axes move at current linear speed. **Speed** is constant between positions.	Spline Linear	Current	SPLINEL *vec pos pos*
Moves TCP through *or near* all consecutive vector positions within a specified range. Axis speed is determined by time definition. **Speed** is constant between positions.		Duration	SPLINEL *vec pos pos time*
Moves axes through *or near* all consecutive vector positions within a specified range. Axes move at current joint speed. **Speed** is constant between positions.	Spline Joint	Current	SPLINE *vec pos pos*
Moves axes through *or near* all consecutive vector positions within a specified range. Axis speed is determined by time definition. **Speed** is constant between positions.		Duration	SPLINE *vec pos pos time*
Moves axes through all consecutive vector positions within a specified range at current joint speed. **Time** is constant between positions.	Spline Time	Current	MOVES *vec pos pos*
Moves axes through all consecutive vector positions within a specified range within time specified. **Time** is constant between positions.		Duration	MOVES *vec pos pos time*

Movement Settings

The Movement Settings dialog box shows the current movement speed and acceleration definitions.



To access this dialog box, open the MOVE dialog box and select **Settings**.

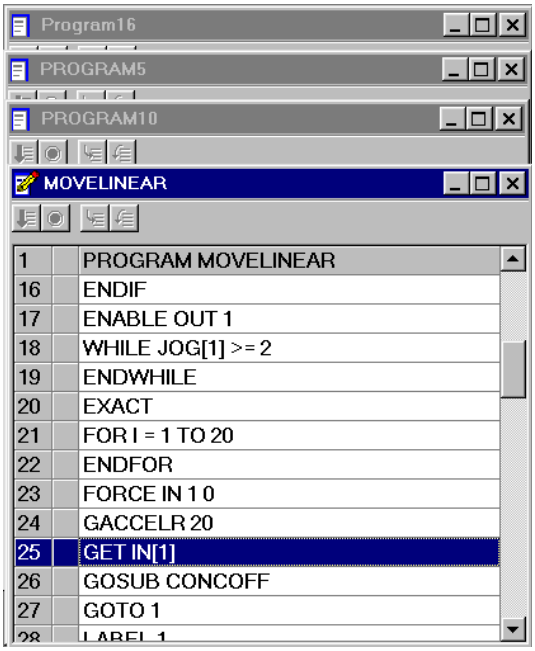
Linear Speed	Speed of robot linear movements. Defined by the **SPEEDL** command. Affects MOVEL(D) and MOVEC(D) and SPLINEL(D) commands.
Joint Speed	Speed of robot joint movements. Defined by the **SPEED** command. Affects MOVE(D), MOVES(D), and SPLINE(D) commands.
Peripheral Speed	Speed of peripheral axis movements. Defined by the **SPEEDP** command.
Direct	Speed of movement set by the last SPEED or SPEEDL command executed in **Direct** mode.
Program	Speed of movement set by the last SPEED or SPEEDL command executed from a running **Program**.
Global Speed Factor	Current setting of global speed factor. Defined by the **GSPEED** command. Applied separately to robot and peripheral axis.
Global Acceleration	Current setting of global acceleration factor. Defined by the **GACCEL** command. Applied separately to robot and peripheral axis.
Exact Mode	Indicates whether the **Exact** mode is enabled or disabled. Applied separately to robot and peripheral axis. When a movement command with D suffix is executed in **Exact** mode, the axes reach the target position with required accuracy.

4

Programs

ACL Program Window

Each Program window displays one ACL program. Many program windows can be opened simultaneously, but only one can be active.



The currently active program is indicated by the “writing pencil” icon in the window’s title bar. When a command dialog box is opened, the pencil icon remains displayed on the program where the command will be entered.

To open a new, empty program, do either of the following:

- Click the New Program button on the ACL-Win main toolbar.
- Select **Program | New Program**. A dialog box allows you to assign a name to the new program.

To open an existing program from the current project, click on the program name in the Project tree.

To open an existing program from another project and include it in the current project, select **Program | Add Program**, and select the program name from the file list.

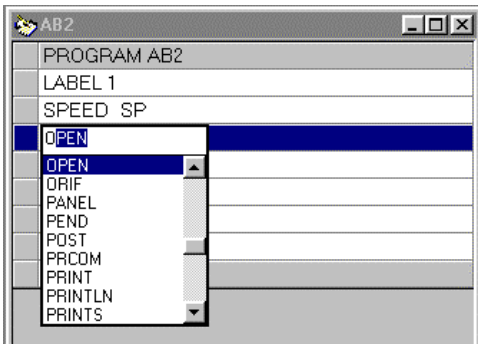
ACL has six reserved program names: AUTO, BACKG, CRASH, EMERG, RECOV and START. Refer to section, “Reserved Program Names,” at the end of this chapter.

Editing and Direct Command Entry

To **write** commands into a program, bring the cursor to the program line at which you want to enter a command, and do any of the following:

- Select a command from the command list.
- Begin typing the first letter(s) of the command from the keyboard. An alphabetical command list will open, and allow you to select the command.
  - Press Enter to open dialog box for the selected command.
  - **Press the spacebar to enable the line editor for direct entry** of command strings.

You can also copy complete command lines from the list of recently used commands. Select **View | History** and double click on the desired command



Command dialog boxes will open whenever required, to allow you to complete command arguments.

To **edit** a command which has already been written, do either of the following:

- Double click on the command line you want to change, or select it and press Enter. The command’s dialog box will open with the fields displaying the arguments used in the selected command.
- Select the command line you want to change **and press the spacebar to enable the line editor**.

Note: Since the dialog boxes for the commands MOVE and HERE remain open until Close is selected, the first click on Program rewrites (overwrites) the current command line, while the second (and subsequent) click on

Program inserts (copies) the edited command at the program cursor's current location.

Program Editing

The Edit menu and icons on the ACL-Win main toolbar provide access to the standard Windows functions and shortcuts for editing program lines.

	Undo	Reverses the last action or operation, such as adding, deleting or modifying a program line. Up to 20 operations can be undone.
	Cut	Cuts the selected code in a Program window and places it on the clipboard.
	Copy	Copies the selected code in a Program window and puts it on the clipboard.
	Paste	Inserts the contents of the clipboard at the current point in the program.
	Find	Performs a search in a Program window.
	Replace	Finds the specified string in the Program window and changes it.

The Program window's toolbar has only a few functions. Right-clicking on the Program window opens its short-cut menu, which provides access to more functions.

Note: A **white** program line must be selected for these functions to be available.

Edit Line	Opens the dialog box for the currently selected command.
Stop	Stops execution of the program.
Run (Program *n*)	Executes the program.
Toggle Breakpoint	Inserts or deletes a breakpoint at the selected line. Breakpoints are used to stop execution of a program at a specific point and allow single-step execution of command lines which follow the breakpoint. Once a breakpoint has been set, the command line is bolded and a red star appears to the left of the command line. When program execution halts at the breakpoint, the command line is highlighted (turns yellow).

When you want to stop program execution at each iteration of a loop, do **not** set the breakpoint at the command which opens the loop (for example, FOR, GOTO, IF, WHILE); instead set the breakpoint at any command within the loop.

Breakpoints can be disabled and enabled. Refer to the section, "Breakpoints," which appears later in this chapter.

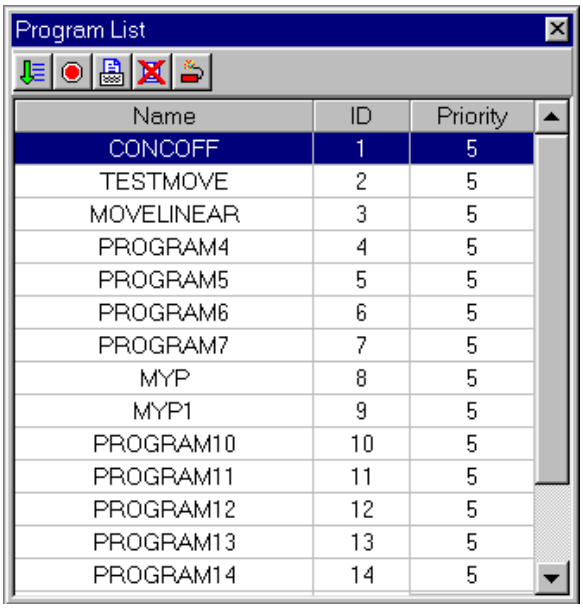
Step Into	Single steps program execution (executes the current command line and stops). When used at a GOSUB command, Step Into opens the called subroutine's Program window, and stops on the subroutine's first line.
Step Over	Single steps program execution (executes current command line and stops). When used at a GOSUB command, Step Over calls and executes the subroutine in its entirety, and then stops on the command line following the GOSBUB command.
Continue	Resumes normal execution of a program which has been halted at a SUSPEND command or at a breakpoint, or when single-stepping program execution following a breakpoint.
Add Watch	Opens the Variable List dialog box, which allows you to mark the variables to be tracked.
Compile	Checks the syntax of the selected program. Searches for errors, such as FOR commands without ENDFOR, IF without ENDIF, and GOTO without a proper LABEL. After the program is compiled, the Compile Errors window opens, and displays the name of the program and the number of errors (in parentheses) which were found. For each error found, the offending command line is listed. Double click on this line to jump to the command within the program.
New Variable	Opens the Define Variable dialog box for naming and defining a new variable.
New Position	Opens the Define Position dialog box for naming and defining a new position.

Toggle Breakpoint, Step Into, Step Over and Continue are enabled only if the program is currently stopped at a breakpoint.

ACL Program List

The Program List displays a list of all programs currently **loaded** in the controller. This list is not available when operating offline.

To display the list of programs, select **View | Programs**.



The Program List dialog box is used to control program execution, and to delete programs from the controller.

The Program List displays the following information:

Name	Name of the program.
ID	A program identification number. This number is automatically assigned by the system, but you can change it. (Select the program in the Project List window and simply typ a new number.) This number is needed for accessing programs from the teach pendant.
Priority	Shows the execution priority value of the program. Value ranges from 1–10, with 10 the highest priority. By default, all programs are assigned a priority of 5 when the controller is turned on. Refer to the command PRIORITY.

The short-cut buttons in this window provide the following functions:

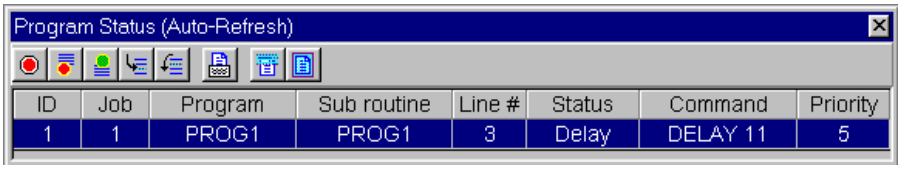
	Run	Runs the currently selected program.
	Abort	Stops all running programs.
	Edit	Displays the Program window for selected program.
	Delete	Removes the currently selected program from controller memory.
	Trigger	Opens the TRIGGER command dialog box.

ACL Program Status

The Program Status window is used to track every **job** (task) currently **running** in the controller. This list is not available when operating offline.

This window is recommended for program debugging since it allows you to verify which jobs are running. It allows you, for example, to step into or over the correct instance of a program which may have been called by two different programs.

To display the list of jobs currently running, select **View | Program Status.**



Note: Since the information displayed in this window changes so rapidly, the Program Status window is updated according to the setting of the Refresh rate. As a result, you may not always see every program line displayed as it is being executed.

This window is available only when the system is operating online.

The Program Status window displays the following information:

ID	The identification number of the program which activated the job.
Job	A system-defined identification number for the job (task).
Program	The name of the program currently being executed.

Subroutine	The name of the subroutine currently being executed. This may be a program called by a GOSUB command, and is not necessarily the program whose ID number is displayed.
Line #	The number of the line in Program currently being executed.
Status	Shows the current status of the job. • Run • Delay (includes instances of a program waiting for a movement command to be completed. • Breakpoint Stop
Command	The command currently being executed.
Priority	Shows the execution priority value of the Job. Value ranges from 1–10, with 10 the highest priority. Refer to the command PRIORITY.

The short-cut buttons in this window provide the following functions:

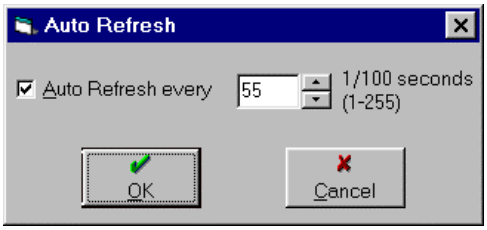
	Stop	Stops the currently selected program.
	Continue	Resumes normal execution of a program which has been halted at a SUSPEND command or at a breakpoint, or when single-stepping program execution following a breakpoint.
	Step Into	Single steps program execution (executes the current command line and stops). When used at a GOSUB command, Step Into opens the called subroutine's Program window, and stops on the subroutine's first line.
	Step Over	Single steps program execution (executes current command line and stops). When used at a GOSUB command, Step Over calls and executes the subroutine in its entirety, and then stops on the command line following the GOSBUB command.
	Suspend	Suspends the currently selected program.
	Edit	Activates the Program window for the selected program.
	Refresh	Refreshes the selected line in the Program Status list. To be used when the Auto-Refresh option is disabled.

Refresh All

Refreshes all lines in the Program Status list. To be used when the Auto-Refresh option is disabled.

Auto Refresh

Right click on the Program Status window to access the Auto Refresh dialog box.



Some data changes so rapidly in the controller that it cannot be displayed or viewed. The Auto Refresh option allows you to define the frequency at which the information displayed in the Program Status dialog box is updated.

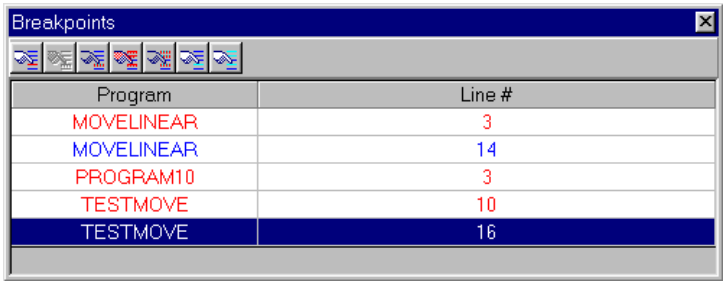
The refresh rate can be defined in the range 1–255 hundredths of a second.

Breakpoints List

Breakpoints are used to stop execution of a program at a specific point and allow single-step execution of command lines which follow the breakpoint.

The Breakpoints window displays a list of all defined breakpoints in all programs listed in current project.

To display the list of breakpoints, select **View | Breakpoints**.



Breakpoints which are defined but disabled are displayed in blue text.

The Breakpoints List displays the following information:

Program

Name of the program containing the breakpoint.

Line #

Line at which breakpoint is set.

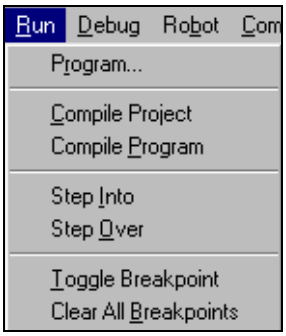
The Breakpoints dialog box has the following options:

Edit	Displays the line with the defined breakpoint in a program code window.
Enable	Enables the selected breakpoint.
Disable	Disables the selected breakpoint. The breakpoint will be ignored during program execution.
Enable All	Enables all defined breakpoints.
Disable All	Disables all defined breakpoints. All breakpoints will be ignored during program execution.
Clear	Deletes the selected breakpoint.
Clear All	Deletes all defined breakpoints. Before resuming normal execution or robotic application, make sure all breakpoints are cleared.

For more information on setting breakpoints, refer to the section, “Program Editing,” earlier in this chapter.

Run Menu

The Run menu allows you to execute and test ACL programs.



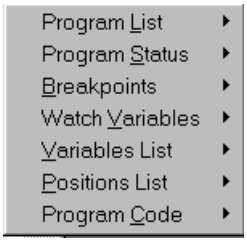
Program	Opens the RUN command dialog box for selecting a program to be executed.
Compile Project	Validates the project as a whole, including all programs and inter-program rules.
Compile Program	Validates the code in the currently active program window.

Step Into	Single steps program execution (executes the current command line and stops). When used at a GOSUB command, Step Into opens the called subroutine's Program window, and stops on the subroutine's first line.
Step Over	Single steps program execution (executes current command line and stops). When used at a GOSUB command, Step Over calls and executes the subroutine in its entirety, and then stops on the command line following the GOSBUB command.
Toggle Breakpoint	Toggles the breakpoint on and off at the current line of the active program window. If there is already a breakpoint defined at the current line of code, it is disabled. If there is no breakpoint at the current line, one is defined.
Clear All Breakpoints	Clears all currently defined breakpoints, in all programs in the active project.

For more information on these functions, refer to the section, "Program Editing," earlier in this chapter.

Shortcuts Menu

The Shortcuts menu provides **keyboard** access to the popup menus of currently open lists and windows. This menu is useful when a pointing device or mouse is not available.



Reserved Program Names

ACL has six reserved names for user programs: AUTO, BACKG, CRASH, EMERG, RECOV and START. You can open and edit these programs 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 after being shut down while in COFF. (See program RECOV below.)

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.

To run BACKG after it has been edited, press and release the emergency button.

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.

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.

Remember! Releasing the EMERGENCY button automatically activates BACKG.

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
```

RECOV

The RECOV program is automatically executed when the controller is powered on or reset after being shut down while in CON(similar to AUTO which runs after a shutdown in COFF).

This program can be used to execute special recovery routines if a power failure occurs during robot operation.

START

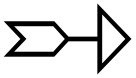
The START program is automatically executed when the Start push button on the controller is pressed, or when a remote Start switch is activated.

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.



5

Commands

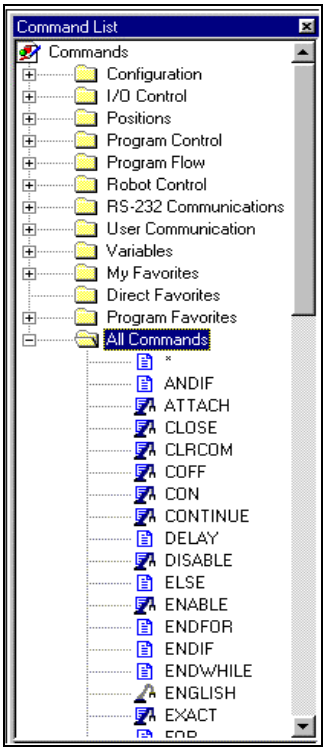
Types of Commands

ACL-Win has two types of commands.

- **Direct** commands, which are executed immediately if the system is operating online.
- **Program** commands, which are executed only during the running of the programs and routines in which they are used.

Some commands are available as both Direct and Program commands.

The **command list** provides access to all ACL commands. The command list contains a number of folders. Double clicking expands and contracts these folders.



- The first set of folders are fixed groups of commands arranged according to function.
- The second set of folders are available for the user to define “favorite” groups for commands. Selecting and dragging commands from other folders copies the commands into the favorite folder. Commands deleted from a favorite folder remain available in the other folders.
- The last folder is a complete listing of all ACL commands, arranged alphabetically.

Each command is marked by one of three icons, which indicate whether the command is a Program and/or Direct command:

Program (edit) command.

Direct command.

Available as **both Direct** and **Program** commands.

Tool tips at the bottom of the command provide a brief explanation of the selected command and examples of command syntax.

Double-clicking on a command usually opens a dialog box, in which you specify required and/or optional parameters for the particular command. If no parameters are required, the command is simply executed or written to the program.

Dialog boxes contain Direct and/or Program buttons to allow you to select how the command is used. These buttons remain disabled until you have entered all required parameter fields.

- The **Direct** button will remain disabled if the software is operating offline.
- The **Program** button will remain disabled if a Program window is not open.

Notations

This chapter presents the ACL commands in alphabetical order.

Each entry includes the command dialog box and the command code format, a description of the command, examples of use and additional notes, including references to related commands and subjects.

The fields in the ACL-Win command dialog boxes are clearly labeled to indicate the type of data which you need to enter, such as “position,”

"variable/constant," "number," and so on. Many fields have a list of options from which you can select.

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

{ } Curly braces enclose a list from which an item *must* be selected.

( ) Parentheses enclose optional items.

/ A slash separates options. For example, ATTACH OFF{R/P} means ATTACH OFFR or ATTACH OFFP.

italics Italicized abbreviations represent data elements or values which are specified when the command code is entered. The most common items are as follows:

arg	argument (usually a string or variable)
axis	axis
duration	duration (time)
n	number
oper	operator (relationship)
pos	position
prog	program
pvect	position vector
value	value
var	variable

[Ctrl]+A (Abort)

Button: 

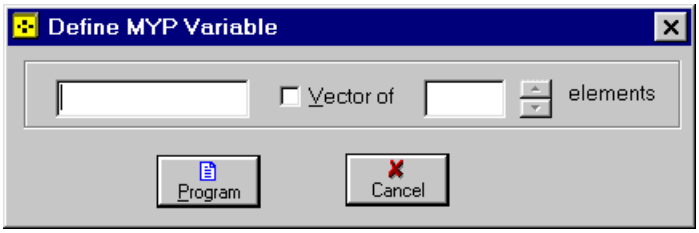
Format: [Ctrl]+A

Description: Immediately aborts all running programs and stops movement of axes.

[Ctrl]+A is enabled as long as ACL-Win is online, even if another software application is currently active.

The [Ctrl]+A abort function in ACL-Win overrides this key combination's function in all other program

#LOCAL



Format: #LOCAL *var* [*n*]

Where: *var* is a variable name;
n is the number of elements in the variable vector

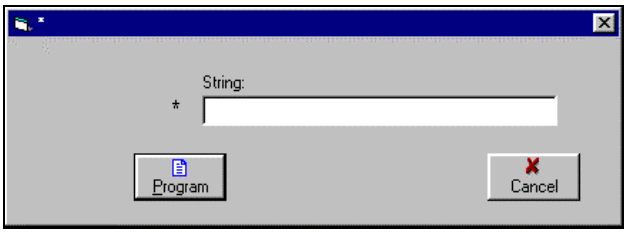
Description: Defines a local variable. A local variable is recognized only by the program in which it is defined.

The title bar of the command dialog box indicates the currently active program.

Example:

■	#LOCAL VAR[10]	Defines a local variable vector of 10 elements.

*



Format: * user comment

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

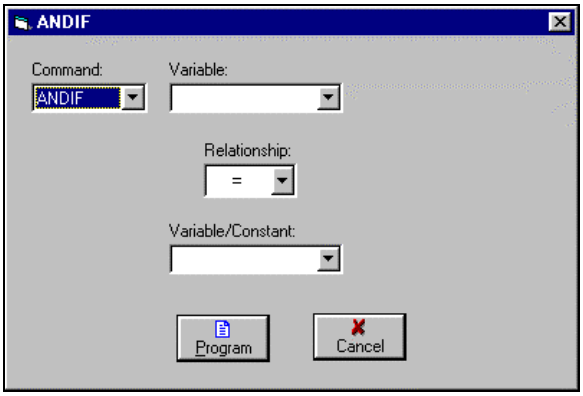
Description: Allows you to annotate your programs.

The * character precedes textual comments within your program.

These comments are not displayed during program execution.

Example: ■ *THIS IS AN EXAMPLE OF A COMMENT

ANDIF



Format: ANDIF *var*1 *oper var*2

Where: *var*1 and *var*2 are variables or constants;
oper can be: <, >, =, <=, >=, <>

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

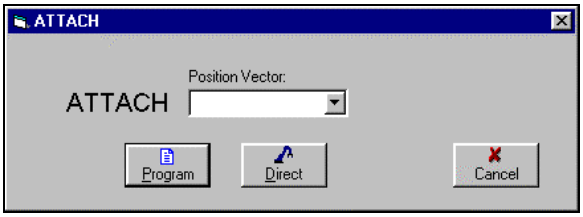
Example: ■

```
IF A=B
    ANDIF C>2
    CLOSE
ELSE
    OPEN
ENDIF
```

If the values of A and B are equal, and if the value of C is greater than 2, close the gripper;
If any of the conditions is not true, open the gripper.*

Note: See also the IF command.

ATTACH / ATTACH OFF



Format:

ATTACH *pvect*

ATTACH OFF(R/P)

Where: *pvect* is a position vector.

Description:

ATTACH *pvect*	Attaches the specified position vector to the teach pendant according to the axis control group for which the position vector is defined. When a vector is attached to the teach pendant, this vector will be the starting vector for the browse selection on the teach pendant. Only one vector at a time may be attached to each group. Attaching another position vector cancels the previous attachment for this group.
ATTACH OFFR	Detaches the robot position vector from the teach pendant.
ATTACH OFFP	Detaches the peripheral axis position vector from the teach pendant..

Example: ■

```
(Define ALPHA[20])
ATTACH ALPHA
```

Define a position vector for the robot named ALPHA containing 20 positions.

BOXOUT

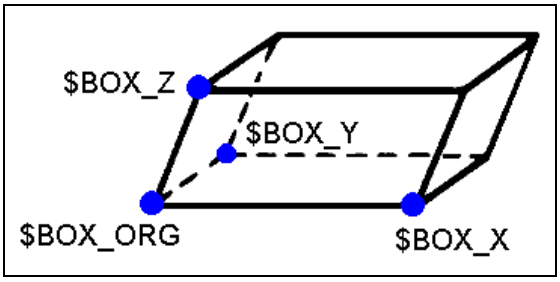
Direct and **Program** command.

Format: BOXOUT {ON / OFF}

Description:

BOXOUT ON	Activates function which calculates shared work space area, and produces controller output response.
BOXOUT OFF	Disables BOXOUT function.

This command checks whether the robot's TCP has entered a workspace shared by two or more robots. This common area is a paralleliped defined by the four predefined system positions $BOX_ORG, $BOX_X, $BOX_Y, $BOX_Z, as shown in the figure below.



When the system determines that a robot TCP has entered the common work area (and to what extent it has penetrated), the controller's response is determined by the BOXOUT command.

BOXOUT ON causes a dedicated controller output to switch on. This is in addition to the BOXSTOP ON response, if it has been enabled.

The output number that is associated with the BOXOUT commands is set in controller parameter 120.

BOXOUT OFF cancels the BOXOUT ON state. It does not cancel the BOXSTOP ON response, if enabled.

Note: See also the BOXSTOP command.

BOXSTOP

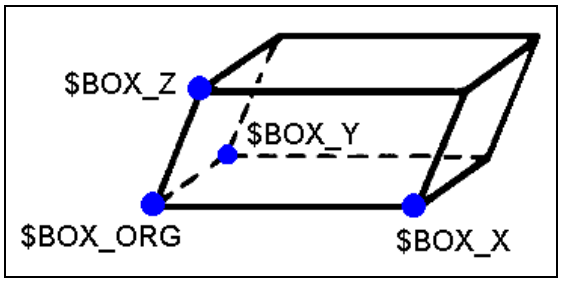
Direct and **Program** command.

Format: BOXSTOP {ON / OFF}

Description:

BOXSTOP ON	Activates function which calculates shared work space area, and causes robot to stop moving.
BOXSTOP OFF	Disables BOXSTOP function.

This command checks whether the robot's TCP has entered a workspace shared by two or more robots. This common area is a paralleliped defined by the four predefined system positions $BOX_ORG, $BOX_X, $BOX_Y, $BOX_Z, as shown in the figure below.



When the system determines that a robot TCP has entered the common work area and to what extent it has penetrated, the robot's response is determined by the BOXSTOP command.

BOXSTOP ON causes the robot to stop moving. This is in addition to the BOXOUT ON response, if it has been enabled.

BOXSTOP OFF cancels the BOXSTOP ON state. It does not cancel the BOXOUT ON response, if enabled.

Note: See also the BOXOUT command.

CLOSE

Format: CLOSE

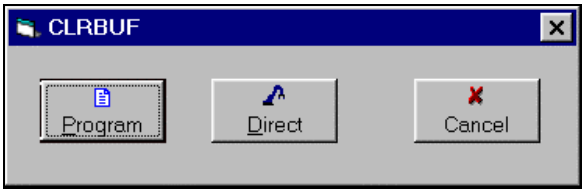
Description: The CLOSE command closes the gripper.

The CLOSE command activates the digital output which controls the gripper.

The number of the output which controls the gripper is defined in Parameter 274. It can also be set in the **Robot | Configuration** menu.

Note: See also the OPEN command.

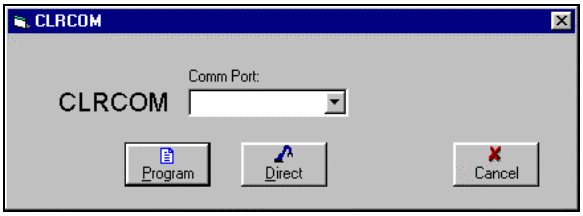
CLRBUF



Format: CLRBUF

Description: 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.

CLRCOM



Format: CLRCOM (*n*)

Where: *n* is an RS232 communication port

Description:

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

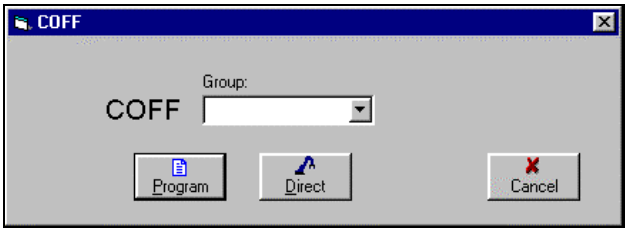
This command can be used to reset the communication ports when an error, such as XOFF without a subsequent XON, interrupts or halts RS232 communication.

Example:

■	CLRCOM 0	Clears the buffer of the controller's RS232 port COM0.

Note: See also the SENDCOM command.

COFF



Button: 

Format: COFF(R/P)

Description:

COFF	Disables servo control of all axes.
COFFR	Disables servo control of robot axes.
COFFP	Disables servo control of peripheral axis.

Servo control is disabled when one of the following occurs:

- 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).

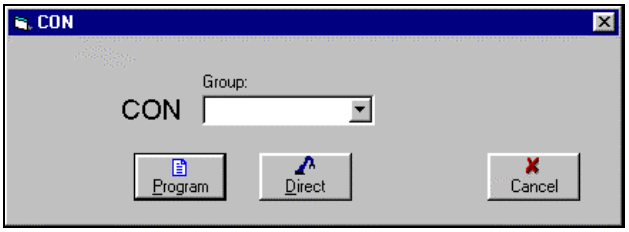
Once COFF has been entered, the axes cannot be operated and the following message appears on both the computer screen and the teach pendant display:

AXIS DISABLED

You must activate CON before motion can resume.

Note: See also the CON command.

CON



Button:

Format: CON(R/P)

Description:

CON	Enables servo control of all axes.
CONR	Enables servo control of robot axes.
CONP	Enables servo control of peripheral axis.

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

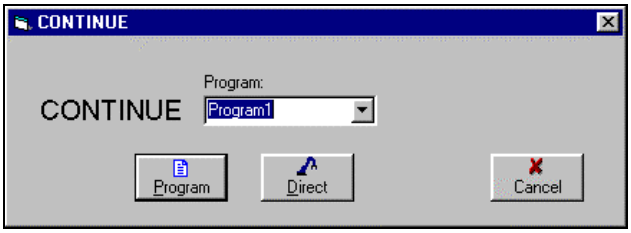
CONTROL ENABLED

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

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

Note: See also the COFF command.

CONTINUE



Format: CONTINUE *prog*

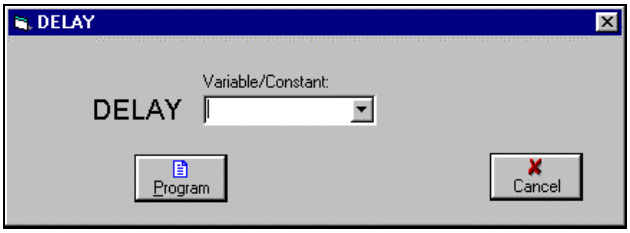
Where: *prog* is a suspended program.

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

Example: ■ CONTINUE ALPHA Resumes execution of program ALPHA.

Note: See also the SUSPEND command.

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).

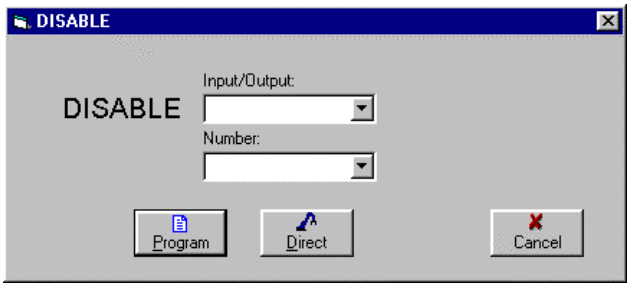
The DELAY command is used for the following purposes:

- 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.

Examples:

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

DISABLE



Format: DISABLE {IN/OUT} *n*

Where: IN is an input; OUT is an output;
n is the I/O index: $1 \le n \le 16$ default
$1 \le n \le 48$ if I/O expansion card installed.

Description:

DISABLE IN *n*
DISABLE OUT *n*

Disconnects the physical input or output from normal system control.

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: See also the ENABLE and FORCE commands.

ELSE

This is a Program command which does not open a dialog box.

Format: ELSE

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

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

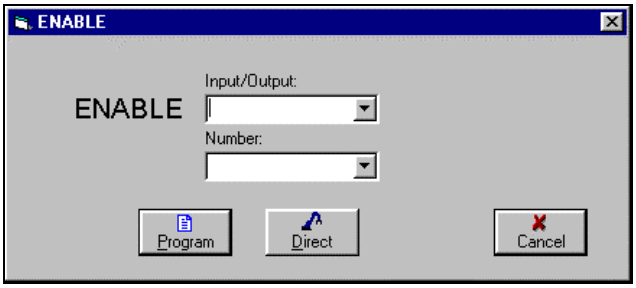
Example: ■

```
IF J>2
    ANDIF A=B
    SET OUT[1]=1
ELSE
    SET OUT[5]=1
ENDIF
```

If the value of J is greater than 2,
and if the values of A and B are equal,
the controller will turn on output1;
If any of the conditions is not true,
the controller will turn on output5.

Note: See also the IF command.

ENABLE



Format: ENABLE {IN/OUT} *n*

Where: IN is an input;
OUT is an output;
n is the I/O index: $1 \le n \le 16$ default
$1 \le n \le 48$ if I/O expansion card installed.

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: See also the DISABLE command.

ENDFOR

This is a Program command which does not open a dialog box.

Format: ENDFOR

Description: Required companion to FOR command.

Ends the block of code 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: See also the FOR command.

ENDIF

This is a Program command which does not open a dialog box.

Format: ENDIF

Description: Required companion to IF command.

Ends the code block to be executed by the IF command.

Example: ■

```
IF XYZ=1
    ANDIF Z[1]=X
    MOVE POS[1]
ELSE
    MOVE POS[2]
ENDIF
```

If the first condition
and the second condition are true,
execute the move;
otherwise,
execute a different move.

Note: See also the IF command.

ENDWHILE

This is a Program command which does not open a dialog box.

Format: ENDWHILE

Description: Required companion to WHILE command.

Ends the block of code to be executed by the WHILE command.

Note: See also the WHILE command.

ENGLISH

This is a Direct command which does not open a dialog box.

Format: ENGLISH

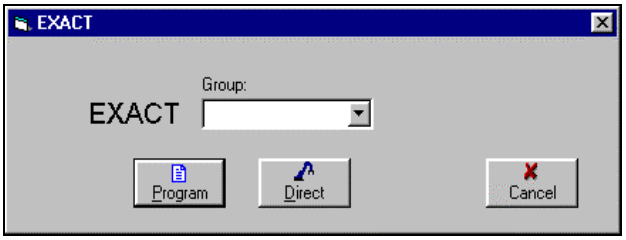
Description: Causes the controller messages to be displayed on the teach pendant in English.

The language definition is saved when the controller is switched off.

The default language is defined as English.

Note: See also the JAPANESE command.

EXACT



Format:

EXACT (R/P)

EXACT OFF(R/P)

Description:

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

MOVED	MOVELD	SPLINED
MOVESD	MOVECD	SPLINELD

The **Exact** and **Exact Off** modes can be applied separately to the robot and the peripheral axes, or to all axes.

EXACT (R/P) — Enables the EXACT mode for all axes, the robot axes only, or the peripheral axis only.

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(R/P) — Disables the EXACT mode for all axes, the robot axes only, or the peripheral axis only.

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 axes are in EXACT mode.

Examples:

- EXACT — EXACT on for the axes.
- EXACT OFF — EXACT off for the robot axes.

- EXACT OFF
  MOVED POS1 500
  MOVED POS2 500

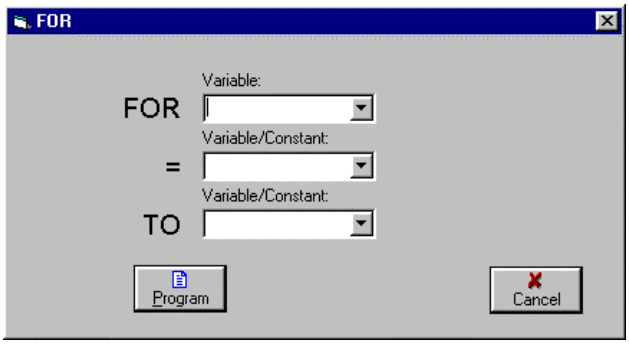
  Axes reach POS1 and POS2 in 5 seconds.

- EXACT
  MOVED POS3

  Axes reach POS3 with required accuracy, regardless of duration.

Note: See also the commands MOVED, MOVESD, MOVELD, SPLINED, SPLINELD, and MOVECD.

FOR



Format: FOR *var*1=*var*2 TO *var*3

Where: *var*1 is a variable;
*var*2 and *var*3 are variables or constants.

Description: Executes a block of code for all values of *var*1, beginning with *var*2 and ending with *var*3.

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

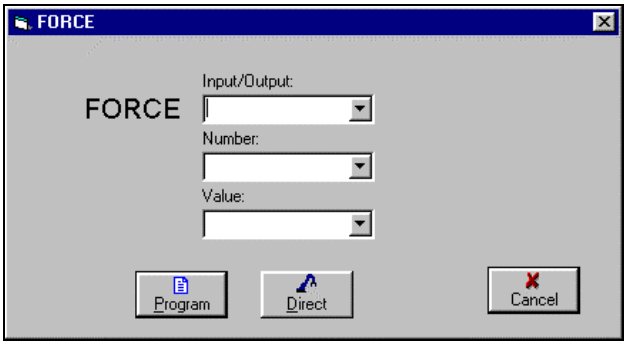
Examples:

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

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

Note: See also the ENDFOR and WHILE commands.

FORCE



Format: FORCE {IN/OUT} *n* {0/1}

Where: IN is an input; OUT is an output;
n is the I/O index: $1 \le n \le 16$ default
$1 \le n \le 48$ if I/O expansion card installed
0=off; 1=on

Description:

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

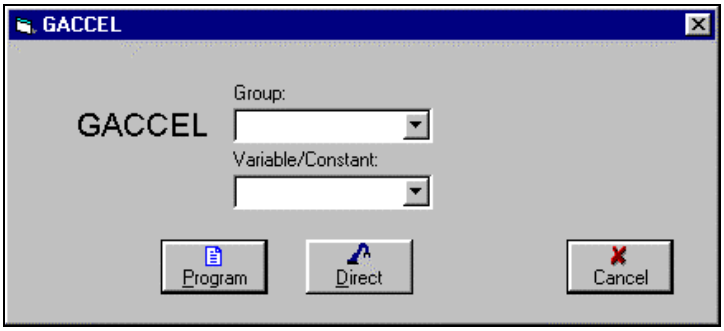
This command is operative only for I/Os which have been disabled by the DISABLE command.

Examples:

- DISABLE IN 5
  FORCE IN 5 1
  — Forces the disabled input 5 to ON state.
- DISABLE OUT 11
  FORCE OUT 11 0
  — Forces the disabled output 11 to OFF state.

Note: See also the DISABLE command.

GACCEL



Format: GACCEL(R/P) *var*

Where: *var* is a variable or constant, whose value is in the range [1...100]

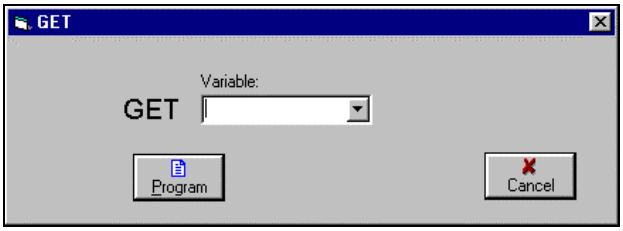
Description: This command applies a global acceleration factor to all movements of all joints, making the maximum acceleration used in movements less than the maximum acceleration allowed for the arm.

This command is useful for: allowing a payload which is heavier than rated for the robot; creating smoother movements; increasing the smothing (rounding of corners) in spline movements.

GACCEL *var*	Sets the acceleration factor of all axes.
GACCELR *var*	Sets the acceleration factor of robot axes.
GACCELP *var*	Sets the accleration factor of the peripheral axis.

GACCEL can be issued at any time, whether or not the robot arm is moving, and takes effect immediately. If a GACCEL command is issued while the robot is in motion, the robot will immediately move according to the new acceleration.

GET



Format: GET *var*

Where: *var* is a user variable.

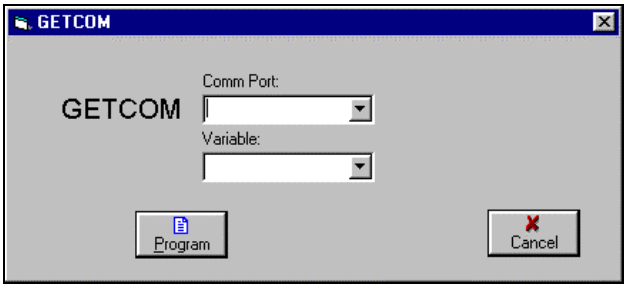
Description: When the program encounters a GET command, it pauses and waits for a keyboard character to be pressed. The variable is assigned the ASCII value of the character that is pressed.

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=*
    ORIF VP=113                 (81 is ASCII for Q)
    RUN Q                       (113 is ASCII for q)
ENDIF
IF VP=82
    ORIF VP=114
    RUN R                       (82 is ASCII for R)
ENDIF                           (114 is ASCII for r)
```

Note: See also the READ command.

GETCOM



Format: GETCOM *n var*

Where: *n* is an RS232 communication port
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
**********************
LABEL1
GETCOM 1 RCV
PRINTLN "RECEIVING ASCII CODE  :  RCV"
GOTO 1
END
```

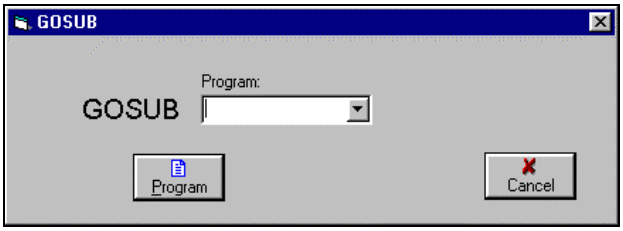
This program waits for a character to be received on RS232 port COM1, and then displays its value on the screen.

If the character A (ASCII 65) is pressed, the following is displayed on the screen:

RECEIVING ASCII CODE : 65

Note: See also the SENDCOM command.

GOSUB



Format: GOSUB *prog*

Where: *prog* is a user program.

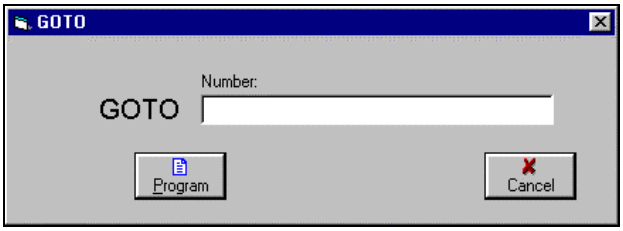
Description: Transfers program control from the main program to *prog*, starting at the first line of *prog*. When the END command in *prog* is reached, execution of the main program resumes with the command which follows the GOSUB command.

Example:

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

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

GOTO



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.

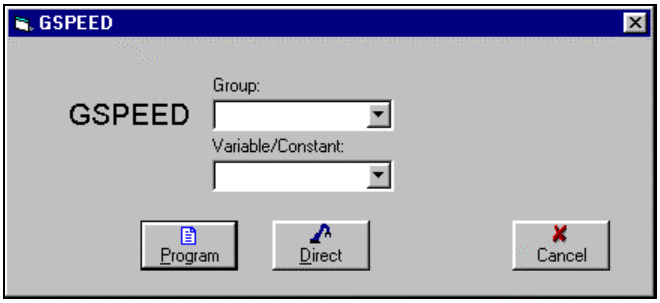
Examples:

■
```
LABEL 5
MOVE POS13
SET A=B+C
GOSUB MAT
GOTO 5
```
This program is executed in an endless loop unless aborted manually.

■
```
LABEL 6
GOSUB BE
SET K=K+1
IF K<500
    GOTO 6
ENDIF
```
This program is executed 500 times and then stops

Note: See also the LABEL command.

GSPEED



Format: GSPEED(R/P) *var*

Where: *var* is a variable or constant, whose value is in the range [-100...+100]

Description: This command applies a global speed factor to all movements of all joints.

Defines the speed of movements in percentages. Maximum speed is 100; minimum is 1. The default speed is 100..

GSPEED	Sets the speed factor of all axes.
GSPEEDR	Sets the speed factor of the robot axes.
GSPEEDB	Sets the speed factor the peripheral axis

GSPEED can be issued at any time, whether or not the robot arm or accessory axis is moving, and takes effect immediately.

GSPEED can be assigned a negative value. In such instances, the arm will return on the same path to the point at which movement was initiated. At that starting position, the arm will stop and wait until a positive value is assigned to GSPEED.

Starting positions are defined as the following:

- For MOVE(L/C): the point at which movement began.
- For SPLINE(L): the point at which the current segment of the spline began; for Example: if the arm was moving from *n*1 to *n*2, the arm will return to position *n*1 if a negative value is assigned to GSPEED.
- For MOVES: the robot's position at the time the MOVES command was executed.

Examples:

- SPEED 80
  SPEEDL 560
  GSPEED 50

  GSPEED reduces all speeds by 50%: joint speed is thus reduced to 40% and linear speed is reduced to 280 mm/sec.

- MOVED A
  MOVE B

  If a negative value (e.g., GSPEED -50) is issued while the arm is moving to B, the arm will return to A.

- MOVED A
  SPLINE M *n*1 *n*2

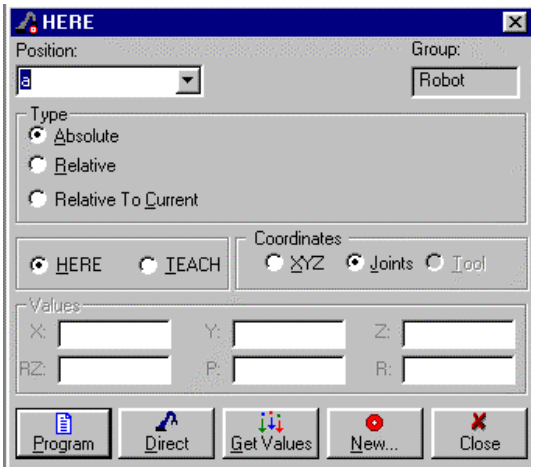
  If a negative value (e.g., GSPEED -50) is issued while the arm is moving from M[*n*] to M[*n*1], the arm will return to M[*n*].

- MOVED A
  MOVES M *n*1 *n*2

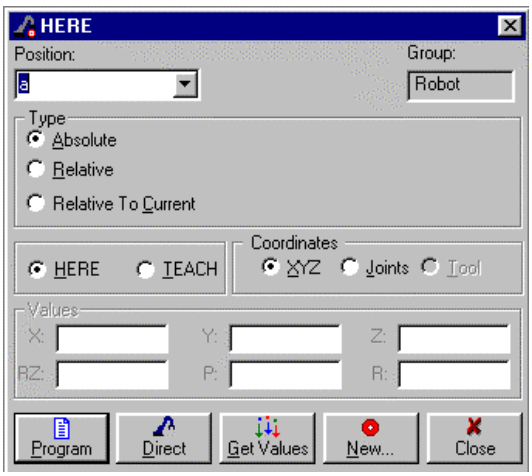
  If a negative value (e.g., GSPEED -50) is issued while the arm is moving from M[*n*] to M[*n*1], the arm will return to A.

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

HERE / HEREC



HERE



HEREC

Button:

Format:

HERE *pos*

Where: *pos* is either a robot or a peripheral axis position.

HEREC pos

Where: *pos* is a robot position.

Description:

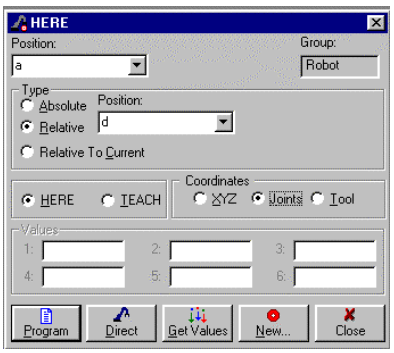
Records an absolute position, according to the current location of the axes.

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

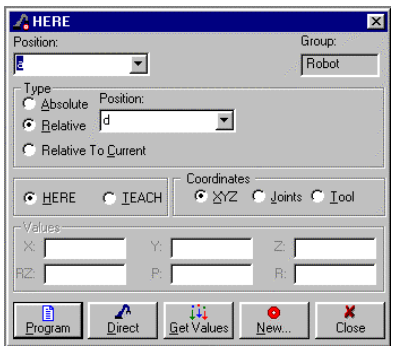
Examples:

HERE POINT	Records the current coordinates for position POINT.
HEREC P[5]	Records XYZ (world) coordinates for position 5 in the vector P.
MOVE POSA 300 DELAY 100 HERE PMID	One second after movement to position POSA begins, position PMID is recorded according to the axes' location at that moment.

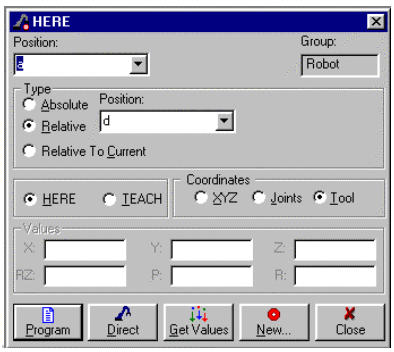
HERER / HERERC / HERERT



HERER



HERERC



HERERT

Button:

Format:

HERER *pos*2 *pos*1
HERERC *pos*2 *pos*1
HERERT *pos*2 *pos*1

Where: *pos*1 is a recorded position for either robot or peripheral axis;
*pos*2 and *pos*1 are defined for the same axis control group.

Description:

HERER *pos*2 *pos*1 allows you to record a position relative to another position.

HERER *pos*2 *pos*1	Records the offset values of *pos*2, relative to *pos*1, in Joint values.
HERERC *pos*2 *pos*1	Records the offset values of *pos*2, relative to *pos*1, in XYZ coordinates.
HERERT *pos*2 *pos*1	Records the offset values of *pos*2, relative to *pos*1, in Tool coordinates.

*Pos*1 must be recorded before *pos*2 can be defined as relative to it. *Pos*2 will always be relative to *pos*1, moving along with and maintaining its offset whenever pos1 is moved.

Examples:

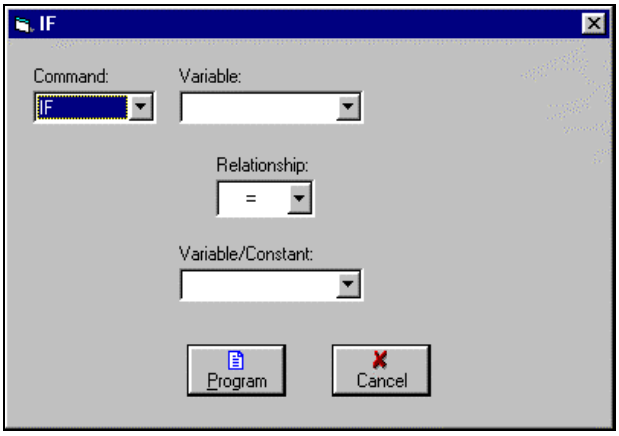
- HERE BB
  (*move robot*)
  HERER AA BB

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

- HERE PLT[1]
  (*move device*)
  HERER PLT[2] PLT[1]

  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.

IF



Format: IF *var*1 *oper var*2

Where: *var*1 is a variable;
*var*2 is a variable or constant;
oper can be: <, >, =, <=, >=, <>

Description: The IF command checks the relation between *var*1 and *var*2 . If it meets the specified conditions, the result is true, and the next sequential program line is executed (code block or command). If it is not true, another code block 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.

■
```
IFIN[3]=1
    SETOUT[7]=1
ELSE
    MOVE10
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: See also the commands ELSE, ANDIF, ORIF, and ENDIF.

JAPANESE

This is a Direct command which does not open a dialog box.

Format: JAPANESE

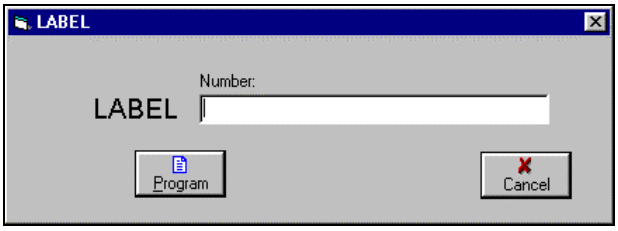
Description: Causes the controller messages on the teach pendant to be displayed in Japanese.

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

The language definition is saved when the controller is switched off.

Note: See also the ENGLISH command.

LABEL



Format: LABEL *labeln*

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

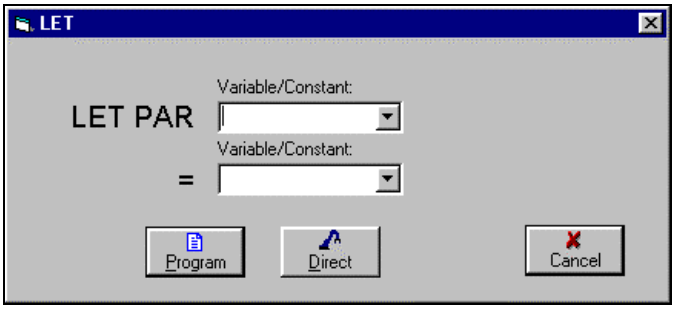
Description: Marks the beginning of a block of code which is executed when the GOTO command is given.

Example: ■

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

Note: See also the GOTO command.

LET PAR



Format: LET PAR *n* = *var*

Where: *n* is a parameter number (variable or constant);
var is a variable or constant.

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

Example: ■ LET PAR 73=9350 Sets the value of parameter 73 to 9350.

Notes:

Warning!
Only experienced users should attempt parameter manipulation.

See also Chapter 7 for information on system parameters, and abide by all warnings given there.

MODULO

Format: MODULO

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

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

MODULO 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 command will be suspended until the robot movement buffer is empty.

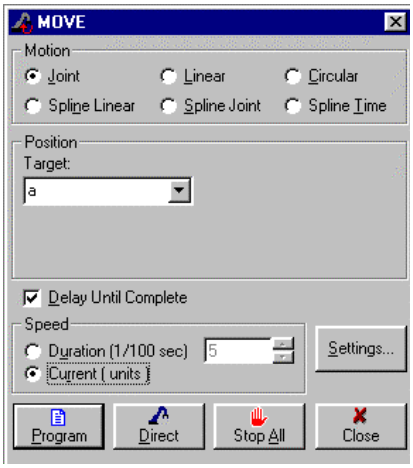
Warning! *Use this command with caution when cables or hoses are connected to the end effector (such as a gripper). Be sure the cables or hoses will not become improperly stretched or entangled following a MODULO command.*

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                       MODULO returns 380 to 20;
SHIFTC POS1 BY R 190         Roll again rotates +190, reaching +210.
MOVE POS1
GOTO  1
```

MOVE / MOVED

Note that execution of MOVE is not synchronized with program flow.



Button:

Format:

MOVE *pos* (*duration*)

MOVED *pos* (*duration*)

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

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 be executed before the intended movement is executed.

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

Duration does not include the acceleration time necessary for the movement. Therefore, the actual movement will always be longer than the duration specified in the command.

Description:

MOVED

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, as shown in the examples below:

```
EXACT OFFA
MOVED POS1 500
MOVED POS2 500
```

Axes reach POS1 and POS2 in 5 seconds.

```
EXACT A
MOVED POS3
```

Axes reach POS3 with required accuracy, regardless of duration.

Whenever the program encounters a movement command with the D suffix (MOVED, MOVECD, MOVELD, MOVESD, SPLINED, SPLINELD), the program is suspended until the arm reaches and stops at the target position.

When the program contains movement commands without the D suffix (MOVE, MOVEC, MOVEL, MOVES, SPLINE, SPLINEL), movements are chained together in smooth sequences, as in SPLINE commands.

- 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 SENT"
```
The robot moves to position 3 and then to position AA. The line COMMAND SENT 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 position3, then to AA and then to POS[1]. Concurrent with the movement to position3, output1 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.

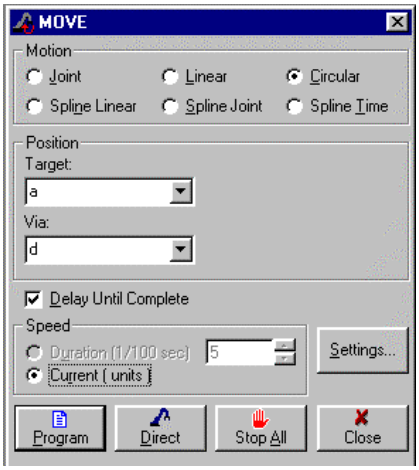
-
```
MOVE 3 500
DELAY 500
MOVE AA 800
DELAY 800
MOVE POS[1] 200
DELAY 200
SET OUT[1]=1
DELAY 1000
```
The robot moves to position 3 in 5seconds, then to AA in 8 seconds, then to POS[1] in 2 seconds. Then output1 is turned on, and a delay of 10 seconds occurs. Total time for program execution is 25 seconds, 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: See also the EXACT command.

MOVEC / MOVECD



Button:

Format: MOVEC *pos*1 *pos*2

MOVECD *pos*1 *pos*2

Description: Moves the robot's TCP (tool center point) along a circular path, from its current *pos*ition to *pos*1, through *pos*2 .

The coordinates of *pos*2 and *pos*1 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, *pos*1, and *pos*2 should define a circle. These three points should not be aligned, and should have different coordinates.

MOVEC/MOVECD is executed in the XYZ (world) coordinate system, and is only valid for robot 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:

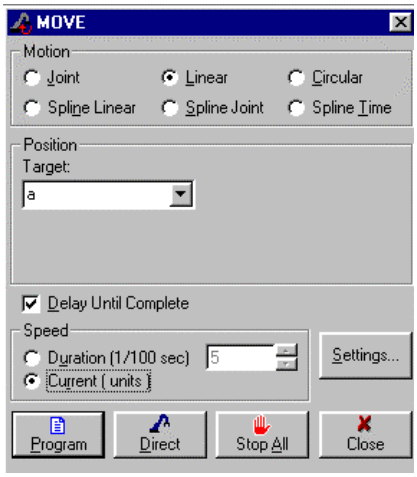
- MOVEC 1 2 — Moves along a circular path from current position to position 1 via position 2.

■ 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: See also the SPEEDL command.

MOVEL / MOVELD



Button:

Format: MOVEL *pos*1 (*duration*)

MOVELD *pos*1 (*duration*)

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.

Duration does not include the acceleration time necessary for the movement. Therefore, the actual movement may be slightly longer than the duration specified in the command.

MOVEL/MOVELD is executed in the XYZ (world) coordinate system, and is only valid for robot axes.

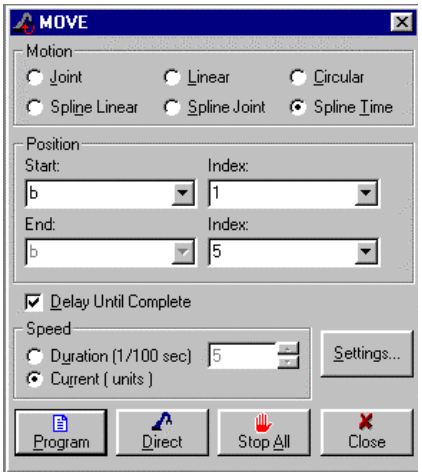
All other aspects of the MOVEL/MOVELD commands are similar to those of the MOVE/ MOVED command.

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: See also the SPEEDL command.

MOVES / MOVESD



Button:

Format:

MOVES *pvect* *n*1 *n*2 (*duration*)

MOVESD *pvect* *n*1 *n*2 (*duration*)

Where: *pvect* is the name of position vector;
*n*1 is the index of the first position;
*n*2 is the index of the last position to be reached.

Description:

Moves the robot axes through any number of consecutive vector positions, from *n*1 to *n*2, without pausing. The trajectory is calculated by a linear interpolation algorthim, then smoothed.

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.

Duration does not include the acceleration time necessary for the movement. Therefore, the actual movement may be slightly longer than the duration specified in the command.

MOVES/MOVESD can be executed only by a robot or multi-axis device.

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

Note: See also the SPLINE and SPLINED commands.

OPEN

Format: OPEN

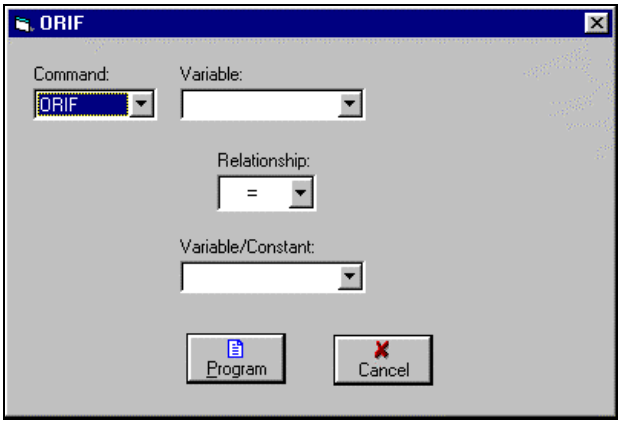
Description: The OPEN command opens the gripper.

The OPEN command activates the digital output which controls the gripper. Parameter 274 defines the number of the output which controls the gripper.

Notes: See also the CLOSE command.

See also the gripper parameters in Chapter 7.

ORIF



Format: ORIF *var*1 *oper var*2

Where: *var*1 and *var*2 are variables or constants;
oper can be: <, >, =, <=, >=, <>

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

Example: ■

```
IF A=B
    ORIF A=D
    CLOSE
ELSE
    OPEN
ENDIF
```

If either A = B
or A = D,
close the gripper;
otherwise,
open the gripper.

Note: See also the IF command.

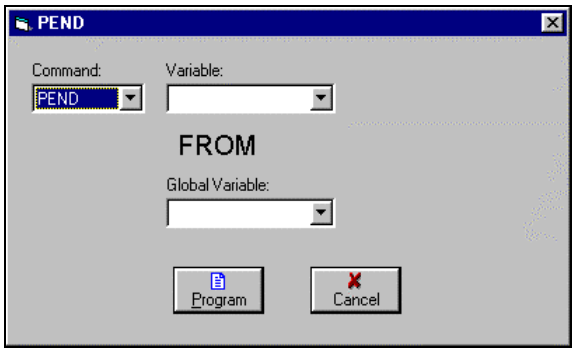
PANEL

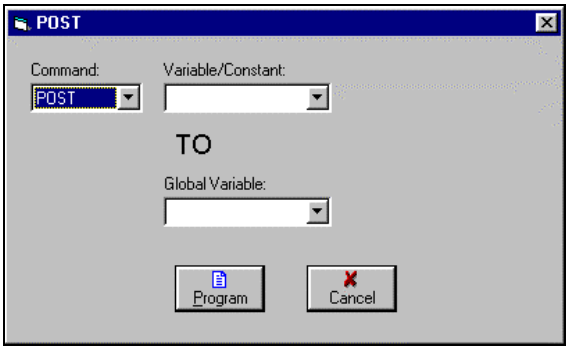
Format: PANEL

Description: Restores (manual) control of the switches and lamps on the robot controller's front panel.

Note: See also the REMOTE command.

PEND / POST





Format:

PEND *var*1 FROM *var*2

POST *var*3 TO *var*2

Where: *var*1 is a variable;
*var*2 is a global variable;
*var*3 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 *var*1 FROM *var*2 command, one of the following occurs:

- If *var*2 has a value of zero, program execution is suspended until another running program sends a non-zero value by means of the POST *var*3 TO *var*2 command.
- If *var*2 has a non-zero value, that value is assigned to *var*1 and the value of *var*2 is set to zero.

Example: ■

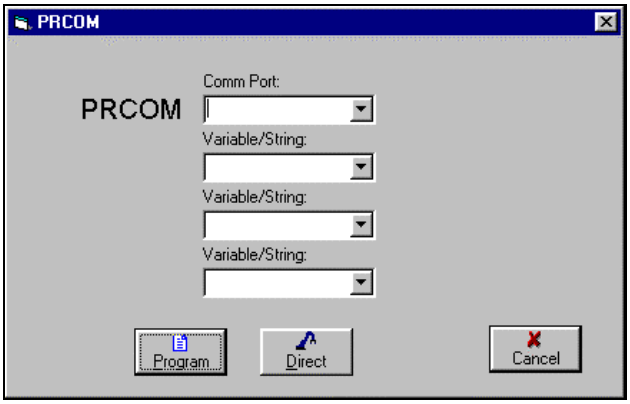
(*Define global variable SIGN*)
(*Define local variable VALUE*)

```
PROGRAM DOACT
**********************
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



Format: PRCOM *n arg*1 (*arg*2 *arg*3)

Where: *n* is an RS232 communication port
arg is a variable or a string within quotation marks ("").

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

Each of the argumnents following PRCOM *n* may contain up to 80 characters and spaces, not including the quotation marks. The text may contain a total of 3 arguments and/or variables.

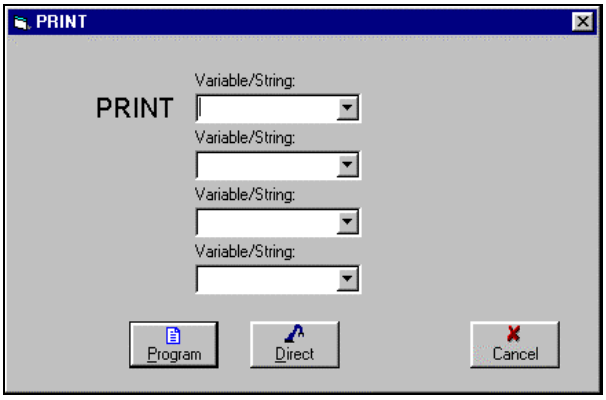
A variable is one argument, regardless of length.

Examples:

- PRCOM 0 "TESTING" — The text TESTING will be transmitted to RS232 port COM0.
- SET X=7
  PRCOM 0 "PRICE IS " X " US$" — The text PRICE IS 7 US$ will be transmitted to RS232 port COM0.

Note: See also the PRLNCOM command.

PRINT



Format: PRINT *arg* (*arg*2 ... *arg*4)

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

Description: Displays strings and variable values on screen.

Right-clicking on the Print display window will display a prompt to clear the display to to save it to a file.

Each of the four arguments following PRINT may contain up to 80 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.

Strings in the PRINT (and PRINTLN and PRINTS) command may include the following control characters:

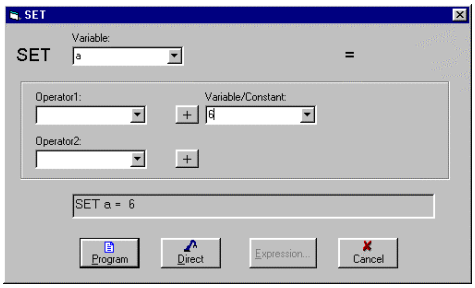
\n	Displays string on new line
\d	Deletes previous character
\1 \2 \3 \4 \5	Character size: **8** [default], 12, 16, 22, 36
\r \g \b **\k**	Character color: red, green, **black** [default]
\c	Clears screen
\\	Displays slash [\]

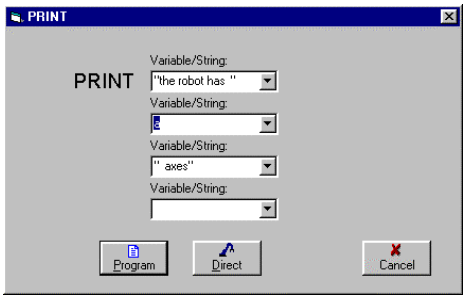
Once changed, character sizes and color remain in effect until the end of the command, or until they are changed by another size or color control character.

At the end of the command all control characters are restored to their default values.

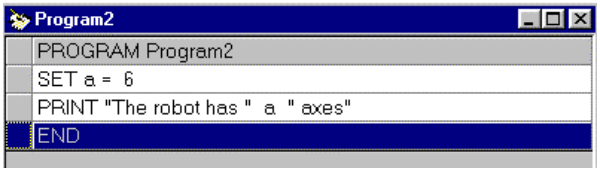
Note: See also the PRINTLN and PRINTS commands.

Examples:

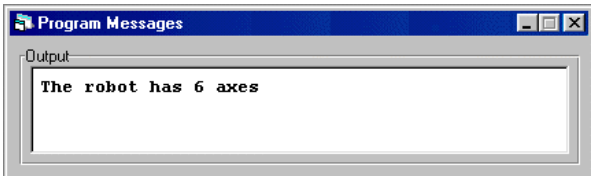


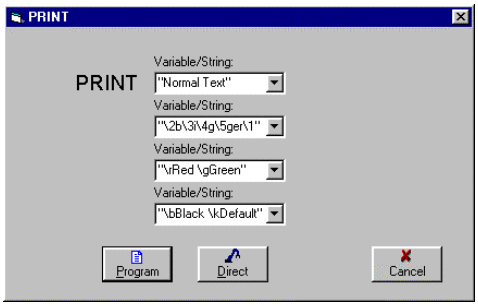


Results in these command lines:

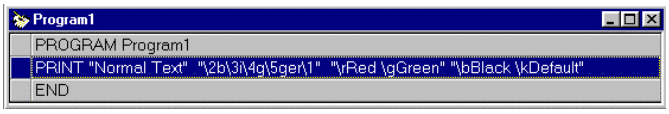


When executed will display:

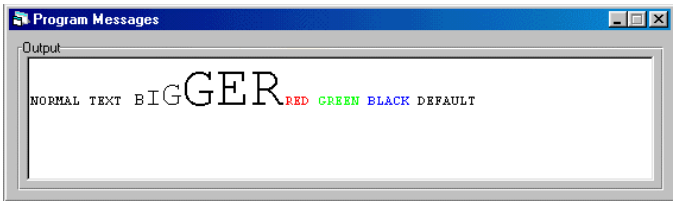




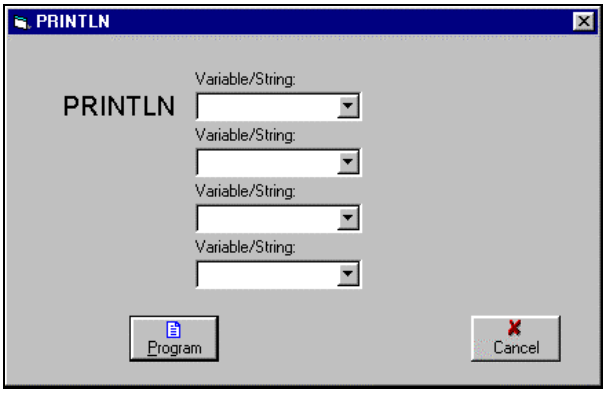
Results in this command line:



When executed, displays this line:



PRINTLN



Format: PRINTLN *arg* (*arg*2 ... *arg*4)

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.

Each of the four arguments following PRINTLN may contain up to 80 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.

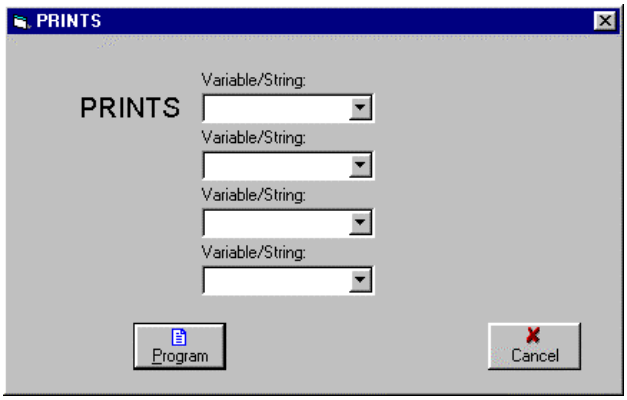
Entering PRINTLN without an argument simply enters a carriage return and a line feed.

Example: ■

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

Note: See also the PRINT command.

PRINTS



Format: PRINTS *arg* (*arg2* ... *arg4*)

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

Description: Displays strings and variable values on screen.

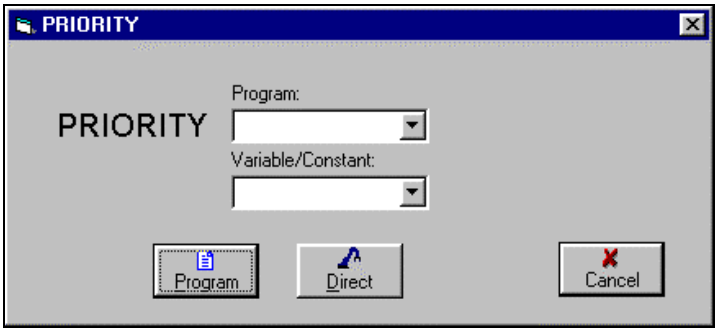
Same as PRINT command, but prints to a program-specific message box.

Each of the four arguments following PRINTS may contain up to 80 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.

Note: See also 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.

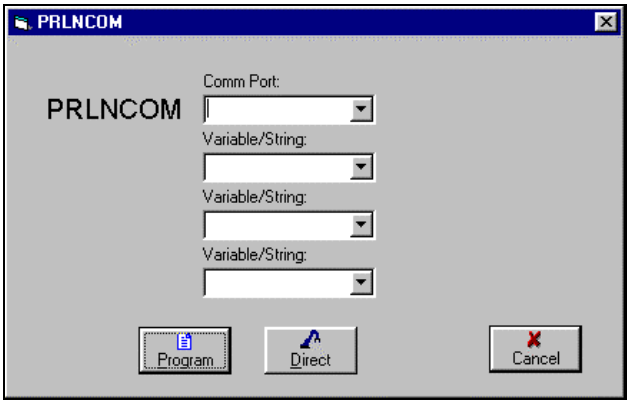
By default (when controller is powered on), all programs are assigned a priority of 5.

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: See also the RUN and DIR commands.

PRLNCOM



Format: PRLNCOM *n arg* (*arg2 arg3*)

Where: *n* is an RS232 communication port;
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.

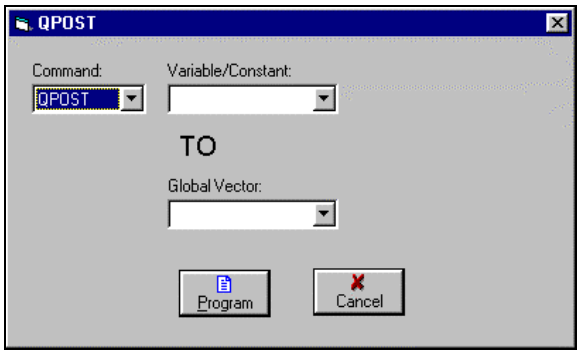
Example: ■ PRLNCOM 0 THE VALUE IS VAL[A]

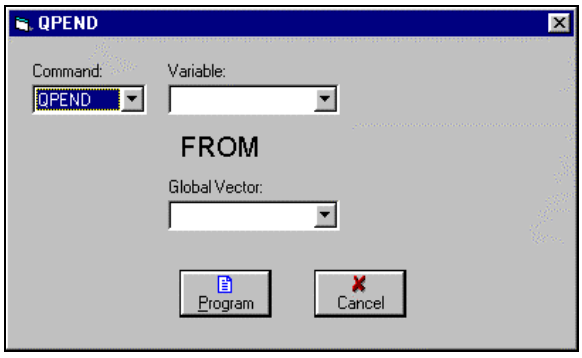
The text THE VALUE IS and the value of variable VAL[A] will be transmitted to RS232 port COM0, followed by a carriage return .

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

Note: See also the PRCOM and READCOM commands.

QPEND / QPOST





Format:

QPEND *var*1 FROM *var*2

QPOST *var*3 TO *var*2

Where: *var*1 is a variable;
*var*2 is a global variable array;
*var*3 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 *var*2 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                          Defines and initializes the queue.
**********************
(Define variable QUEUE[10] )
(Define variable I )
FOR I=1 TO 10
    SET QUEUE[I]=0
ENDFOR
END

                                       Takes a value from a queue.
PROGRAM DOACT                          Program ACT will run when values
**********************                 are deposited in QUEUE by the
```

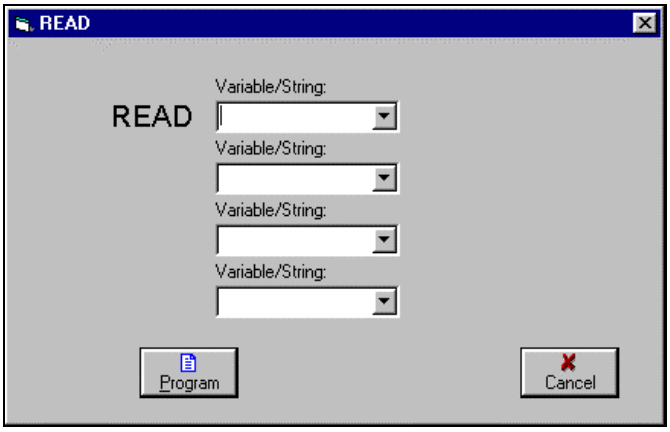
```
(Define variable VALUE)
LABEL 1
QPEND VALUE FROM QUEUE
RUN ACT
GOTO 1
END

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

program SEND. If no value has been sent, DOACT will be suspended until the arrival of a value.

Puts a value in a queue.

READ



Format: READ *arg* (*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 dialog box 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 the dialog box must be a numeric value. Pressing <Enter> without specifying a value will enter a value of 0.

Example:

■ READ "enter value of x" X

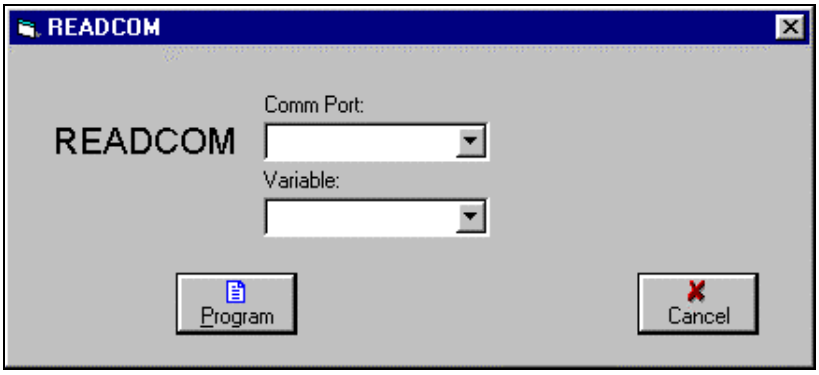
Will display on screen:

enter value of x

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

Note: See also the PRINT command.

READCOM



Format: READCOM *n var*

Where: *n* is an RS232 communication port;
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 0, RPART
IF RPART > 9999
    PRINTLN "CANNOT PRODUCE MORE THAN 9999 PIECES"
```

Note: See also the PRLNCOM command.

REMOTE

Format: REMOTE

Description: By configuring parameters and issuing the command REMOTE, some panel functions (switches, lamps, LEDs) can be operated in Remote mode.

In Remote Mode, control of some of the controller's front panel functions are transferred to external switches and lamps by means of the system's digital inputs and outputs.

REMOTE will affect only the switches and lamps which have been configured for remote operation by parameter definition.

REMOTE will have not have any effect if the controller parameter for the specified switch or light is set to 0.

When set by parameter 15, the Remote HOLD function, remains active regardless of the REMOTE command. This allows the user to chain several safety switches.

Moreover, when the Remote HOLD input is not wired, the state of the Remote HOLD input is defined as active, thus causing the system and robot to immediately enter the HOLD mode. It is is thus necessary to wire the Remote HOLD input to Normally Closed before setting the Remote HOLD parameter.

The status of all lamps and LEDs on the panel (Error, Servo, Run, Hold and Remote) are copied to the Remote output,

The command PANEL cancels the REMOTE command.

Controller Function	I/O Connection	Parameter
Error Reset Switch	Input	PAR 114
START Push Button	Input	PAR 16
Servo ON Switch	Input	PAR 14
Run/Hold Program Switch	Input	PAR 15
Program Running Lamp	Output	PAR 117

Error Indicator Lamp	Output	PAR 115
Servo Lamp	Output	PAR 119
Hold Lamp	Output	PAR 118
Remote LED	Output	PAR 116
Remote HOLD	Input	PAR 15
Remote REMOTE	Input	Par 111

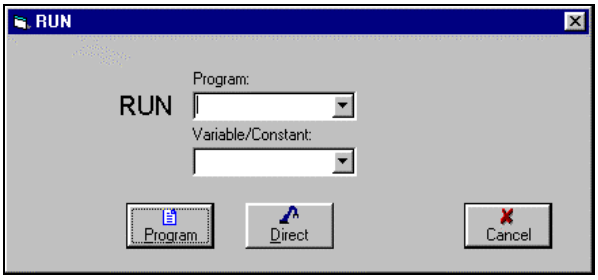
HOLD

Notes:

See also the PANEL and LET PAR commands.

Refer to the Controller-BRC User Manual for more information on controller functions.

RUN



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 PROGRAM 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,

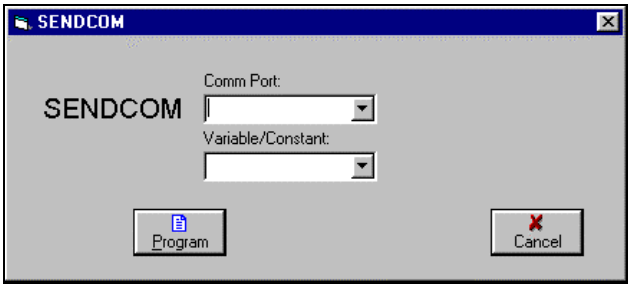
the program's priority is set to the value last defined by a preceding PRIORITY or RUN command.

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: See also the PRIORITY command.

SENDCOM



Format: SENDCOM *n var*

Where: *n* is an RS232 communication port
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.

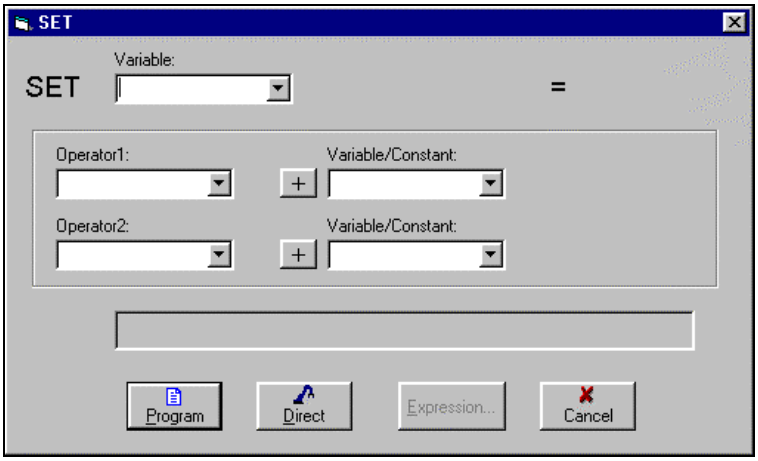
Example:

```
PROGRAM ESC
******************
(Define variable I)
CLRCOM 0
FOR I=1 TO 5
    SENDCOM 2, 27
    DELAY 20
ENDFOR
END
```

This program clears the buffers of RS232 port 0. It then sends 27, the ASCII code for [Esc], five times to port2.

Note: See also the GETCOM command.

SET



Format:

SET *var*1=*var*2

SET *var*1=*oper var*2

SET *var*1=*var*2 *oper var*3

SET *var*1=COMPLEMENT *var*2

SET *var*=PVAL *pos axis*

SET *var*=PVALC *pos coord*

SET *var*=PSTATUS *pos*

SET *var*=PAR *n*

Where: *var* and *var*1 is a variable;
*var*2 and *var*3 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 an XYZ (world) coordinate: X, Y, Z, or P or R;
n is a parameter number.

Description:

1. SET *var*1=*var*2 — Assigns the value of *var*2 to *var*1.

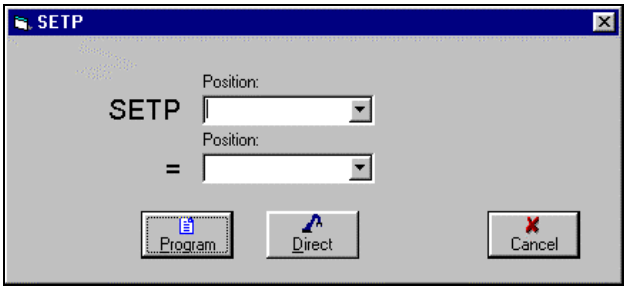
2. SET *var*1=*oper var*2 — The operation is performed on *var*2 and the result is assigned to *var*1 .

	If *oper* is ABS	Assigns the absolute value of *var*2 to *var*1
	If *oper* is NOT	Assigns the logical negative value of *var*2 to *var*1. If *var*2 ≤ 0, *var*1 = 1; If *var*2 >0, *var*1 =0.
3.	SET *var*1=*var*2 oper *var*3	
	If *oper* is : +, -, *, /, MOD	The operation is performed on *var*2 and *var*3 and the bitwise result is assigned to *var*1 .
	If *oper* is: AND, OR	The binary operation is performed on *var*2 and *var*3 and the result is assigned to *var*1.
	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 *var*3 is computed and then multiplied by *var*2. (*Var*2 must be large enough to give the expected accuracy.) The value of *var*3 is an expression of degrees.
	If *oper* is: ATAN, EXP, LOG	In order to use a practical value for *var*3, *var*3 is first divided by 10,000; then the function is applied. The result is then multiplied by *var*2. (The result of the ATAN function is an expression of degrees.)
4.	SET *var*1=COMPLEMENT *var*2	Each individual bit of the binary representation of *var*2 is inverted, and the result is assigned to *var*1 .
5.	SET *var*=PVAL *pos axis*	Assigns *var* the joint value of the specified axis in the specified position. (See also the PVAL command.)
6.	SET *var*=PVALC *pos coord*	Assigns *var* one of the XYZ (world) coordinates of the specified robot position.
7.	SET *var*=PSTATUS *pos*	Assigns *var* a value according to the type of the specified position.
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 JV=PVAL BUF1 1	JV receives the joint coordinate of axis 1 at position BUF1.
SET XC=PVALC POS1 X	XC receives the value of the robot's X coordinate position POS1.
SET C=PVALC POSITION Y	C receives the value of the robot's current Y coordinate.
SET A=PAR 76	The value of parameter 76 is assigned to variable A.
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



Format: SETP *pos*2=*pos*1

Where: *pos*1 is a recorded position;
*pos*1 and *pos*2 are both defined as a robot or a peripheral position.

Description: Copies the coordinate values and position type of *pos*1 to *pos*2.

Both positions are now identical.

If you do not define *pos*2 befor executing this command, the system will automatically define it as a robot position, and assign it a numerical name.

Use the Define Position dialog box to define *pos*2 if you want it to be a peripheral position and/or to have an alphanumeric name.

This command is useful for preparing *pos*2 so that the SETPV command can be used to change one value of that position.

Examples:

- `SETP POINT=PLACE`

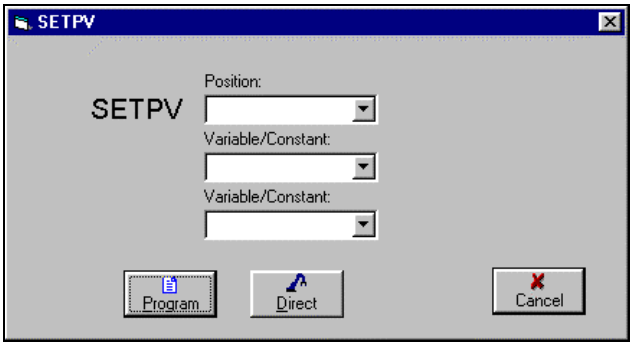
  Position POINT is assigned the coordinate values and type of position PLACE.

- (*Define variable I*)

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

  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.

SETPV



Format: SETPV *pos axis var*

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

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.

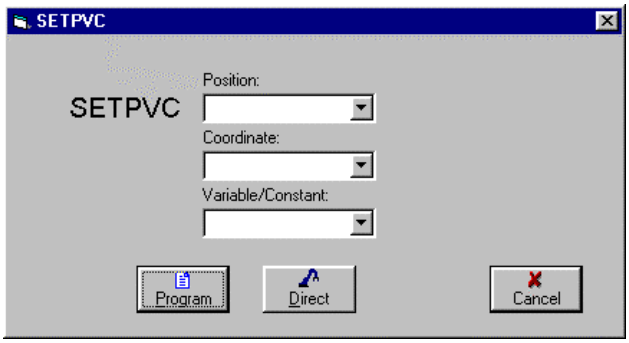
Examples:

- 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.
- SETPV POSITION 3 VARP — The robot will immediately move to the position where joint 3 = VARP

Notes: SETPVC *pos coord var* is the comparable command for changing the value of an XYZ (world) coordinate.

See also the SHIFT, SHIFTC and TEACH commands.

SETPVC



Format: SETPVC *pos coord var*

Where: *pos* is a recorded robot position;
coord is an XYZ (world) coordinate: X, Y, Z, P or R;
var is a variable expressed in microns (X,Y,Z) or millidegrees (pitch, roll).

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

The value of the XYZ (world) coordinate which is modified by this command is defined in microns. Pitch and roll values are defined in millidegrees.

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

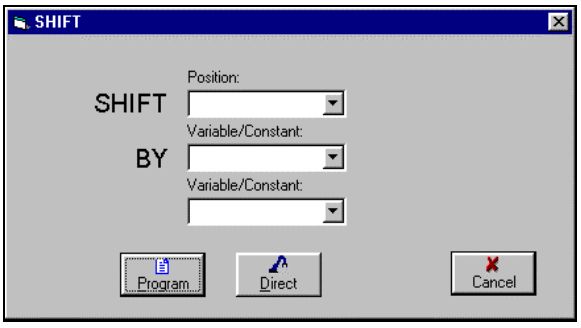
Examples:

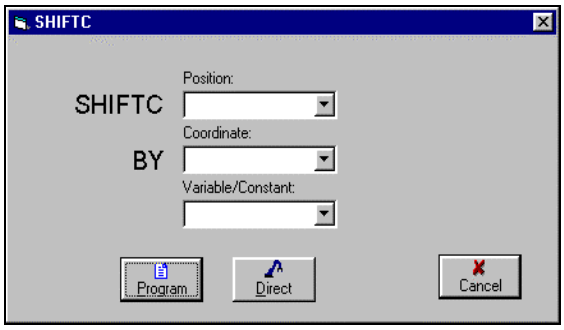
- SET VARA=7000
  SETPVC POSA Y VARA
  The Y coordinate for robot position POSA is changed to 7 millimeters.
- SETPVC POSA Y 7000
  The Y coordinate for robot position POSA is changed to 7 millimeters.
- SETPPA=POSITION
  SETPVCPA X 25000
  SETPVCPA P -45000
  Position PA receives the coordinates values 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.

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

See also the SHIFT, SHIFTC and TEACH commands.

SHIFT / SHIFTC





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 position;
coord is an XYZ (world) coordinate: X, Y, Z, P or R;
var is a variable expressed in microns (X,Y,Z) or millidegrees (P,R).

Description:

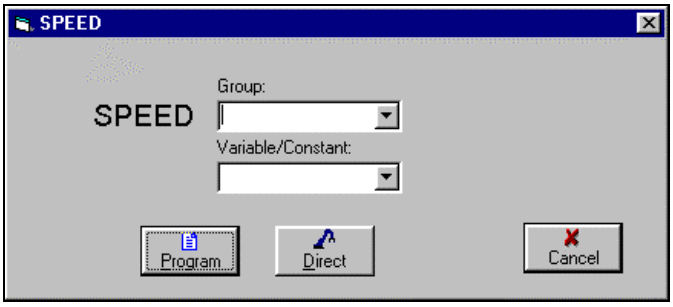
Used for position modification, this command enables you to change the coordinates of a recorded position by an offset value.

SHIFT	Modifies joint coordinates; shifts the position by one joint value.
SHIFTC	Modifies XYZ (world) coordinates; shifts the position by one XYZ coordinate. The value of the XYZ coordinate which is modified by this command is defined in microns. Pitch and roll values are defined in millidegrees.

Examples:

SHIFT P200 BY 1 3000	Robot position P200 is offset by 3000 encoder counts along axis 1.
SHIFTC POS99 BY R 20000	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.

SPEED



Format: SPEED(R/P) *var*

Where: *var* is a variable or constant.

Description: SPEED or SPEEDR sets the speed of the robot axes.

SPEEDP sets the speed of the peripheral axis.

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 amd determines the speed of movement when the MOVE(D), MOVES(D) and Joint SPLINE(D) commands are executed in DIRECT mode.

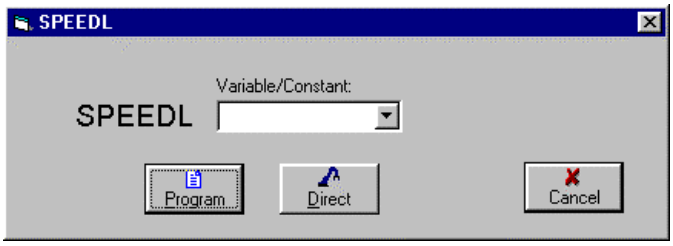
In PROGRAM mode, the SPEED command takes effect after it is executed from within a program. Determines the speed of movement when the MOVE(D), MOVES(D) and Joint SPLINE(D) commands are executed from within a program.

Example:

- SPEED 20 — Sets speed of joint movements of robot to 20% of maximum speed.
- SPEEDR 50 — Sets speed of joint movements of robot to 50% of maximum speed.

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

SPEEDL



Format: SPEEDL *var*

Where: *var* is a variable or constant expressed in mm/sec.

Description: SPEEDL sets the speed of robot 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 and determines the speed of movement when the MOVEL(D), MOVEC(D) and Linear SPLINE commands are executed in DIRECT mode.

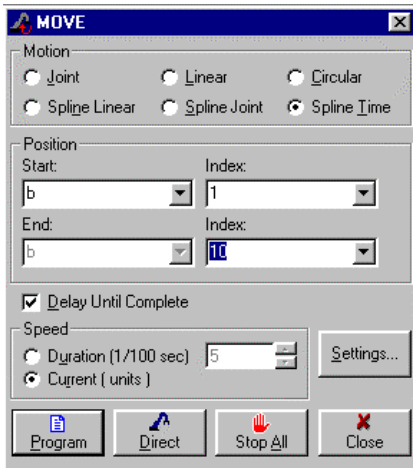
In PROGRAM mode, the SPEEDL command takes effect after it is executed from within a program. Determines the speed of movement when the MOVEL(D), MOVEC(D) and Linear SPLINE commands are executed from within a program.

Example:

- SET VARSP 12
SPEEDL VARSP — Sets speed of linear/circular movements of robot to 12 mm/sec.
- SPEEDL 12 — Sets speed of linear/circular movements of robot to 12 mm/sec.

Note: See also the MOVE(D), MOVES(D), SPLINE(D), SPEED and SHOW SPEED commands.

SPLINE / SPLINED



Button:

Format:

SPLINE *pvect n*1 *n*2 (*duration*)

SPLINED *pvect n*1 *n*2 (*duration*)

Where: *pvect* a position vector;
*n*1 is the index of the first position;
*n*2 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 *n*1 to *n*2, 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).

Positions in the vector may be of any type (joint, XYZ, absolute, relative).

The SPLINE commands generate a smooth path of robot movement through or close to the points of the vector, from *n*1 to *n*2. 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 inthe vector. At high speeds, the trajectory rounds the corners in order to keep acceleration within safe limits.

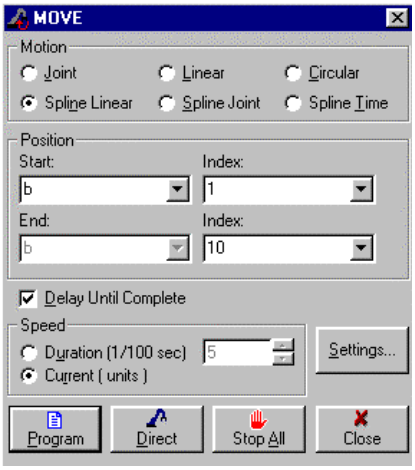
The joint speed of the movement between any two consecutive positions is constant.

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

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.

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

SPLINEL / SPLINELD



Button:

Format:

SPLINEL *pvect n*1 *n*2 (*duration*)

SPLINELD *pvect n*1 *n*2 (*duration*) **Program** command only

Where: *pvect* a position vector;
*n*1 is the index of the first position;
*n*2 is the index of the last position to be reached.

Description:

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

SPLINELD — Same as SPLINEL, except that the command following the SPLINELD command will not begin execution until the robot has reached last position (as in MOVED command).

Positions in the vector may be of any type (joint, XYZ, absolute, relative).

The SPLINEL command is applicable only to robot axes.

The SPLINEL commands generate a smooth path of robot movement through or close to the points of the vector, from *n*1 to *n*2. The trajectory is calculated so that the speed and acceleration are kept within safe limits, according to parameters 536 and 537 (maximum linear and angular speed), and parameters 533 and 534 (maximum linear and angular 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.

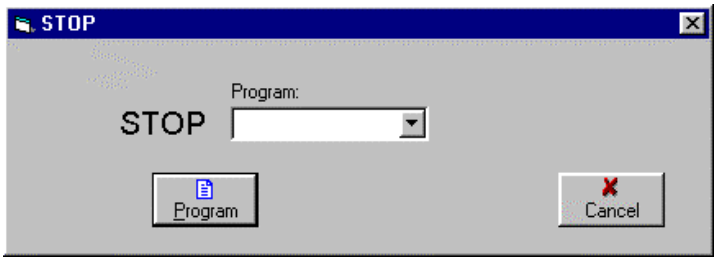
The linear speed of the movement between any two consecutive positions is constant.

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

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 SPLINEL movement.

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

STOP



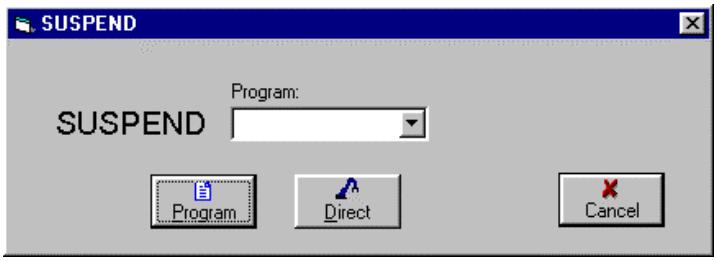
Format: STOP (*prog*)

Description:

STOP	Aborts all programs and all movements.
STOP *prog*	Aborts the running of the specific program only.

Example: ■ STOP DEMO — Aborts program DEMO.

SUSPEND



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: See also the CONTINUE command.

TCP

Format: TCP

Description: The command calculates the tool center point (TCP) based upon five positions which are stored in the predefined system position vector $TCP[5].

You must record the positions in vector $TCP[5] before executing the TCP command. To record these positions, bring the robot's tool center point to the same location five times, with a different orientation each time.

The TCP command calculates the center of the sphere determined by the five positions and the result of the calculations is stored in parameters 311, 312 and 313, respectively the X, Y and Z coordinates of the TCP within the tool frame.

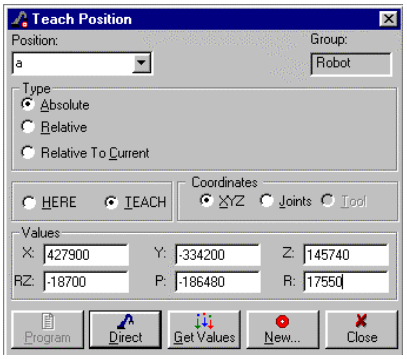
After the TCP command is executed and parameters 311, 312 and 313 are updated, the values of tool parameters 308, 309 and 310 are also updated.

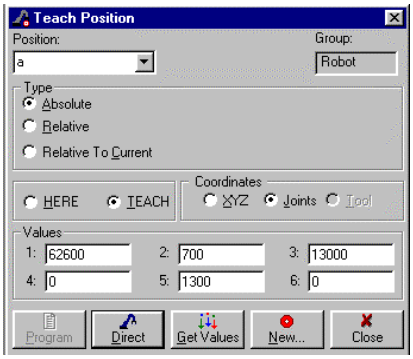
MOVEC, MOVEL, SPLINEL, and teach pendant movement commands in XYZ and Tool are executed according to the robot's TCP.

Notes: See also the TOOL, MOVEC, MOVEL, SPLINEL commands.

Refer to the section "System Positions" in Chapter 3.

TEACH





Button:

Format: TEACH *pos* **Direct** command only

Where: *pos* is a robot position

Description:

Records robot positions according to user defined coordinate values.

If coordinates have already been recorded for the selected position, they are displayed in the Values field. If a coordinate value has not yet been recorded for this position, the field is empty.

X, Y and Z coordinates are defined in microns (1/1000 mm).
Z-Roll, Pitch and Roll (ZR,P, R) values are defined in thousandths of a degree.

Joints coordinates are defined in encoder units.

Depending on the options selected in the TEACH dialog box, different types of positions are recorded.

Absolute robot position recorded in …
- XYZ Coordinates
- Joints Coordinates

Relative to another robot position by …
- XYZ Coordinates
- Joints Coordinates
- Tool Coordinates

Relative to the robot's current position by …
- XYZ Coordinates
- Joints Coordinates
- Tool Coordinates

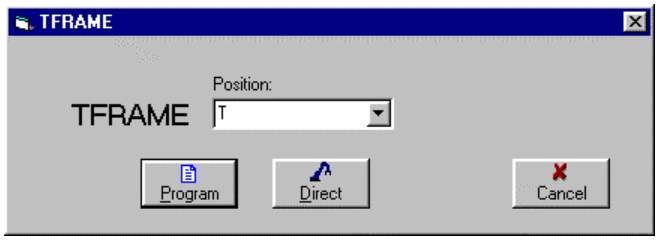
To enable the recording of positions from within a running ACL program, use the commands SETPV and SETPVC for each coordinate that you want to record.

Notes:

See also the HERE, SETPV, SETPVC commands

Refer to the section "Recording Positions" in Chapter 3.

TFRAME



Format:

TFRAME *pos*

Where: *pos* is a robot position

Description:

This command calculates the coordinates of *pos* within the tool frame, according to the coordinates of three predefined system positions, $FRAME_ORG and $FRAME_X and $FRAME_XY.

The coordinates of the *pos* are calculated such that the Tool coordinates at that position will be:

- The Tool X axis is defined by positions $FRAME_ORG and $FRAME_X
- The Tool XY plane is defined by the positions $FRAME_ORG, $FRAME_X and $FRAME_XY.

You must record coordinates for positions $FRAME_ORG and $FRAME_X and $FRAME_XY before executing the TFRAME command.

This TFRAME function allows you to define a reference position (*pos*). This allows the robot to tend different pallets with identical arrangement of objects by means of **Relative by Tool** positions.

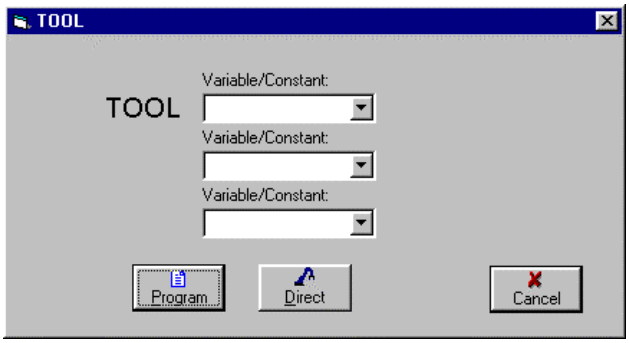
The reference position may be, for example, one corner of the pallet. All other positions are recorded as Relative by Tool to the reference position. To move the robot from one pallet to the next, you need to define the object arrangement only once, and to know only the pallet's reference position.

Notes:

Refer to the section "System Positions" in Chapter 3.

Refer to the section "World (XYZ) and Tool Coordinate Systems in Chapter 3.

TOOL



Format: TOOL *length offset angle*

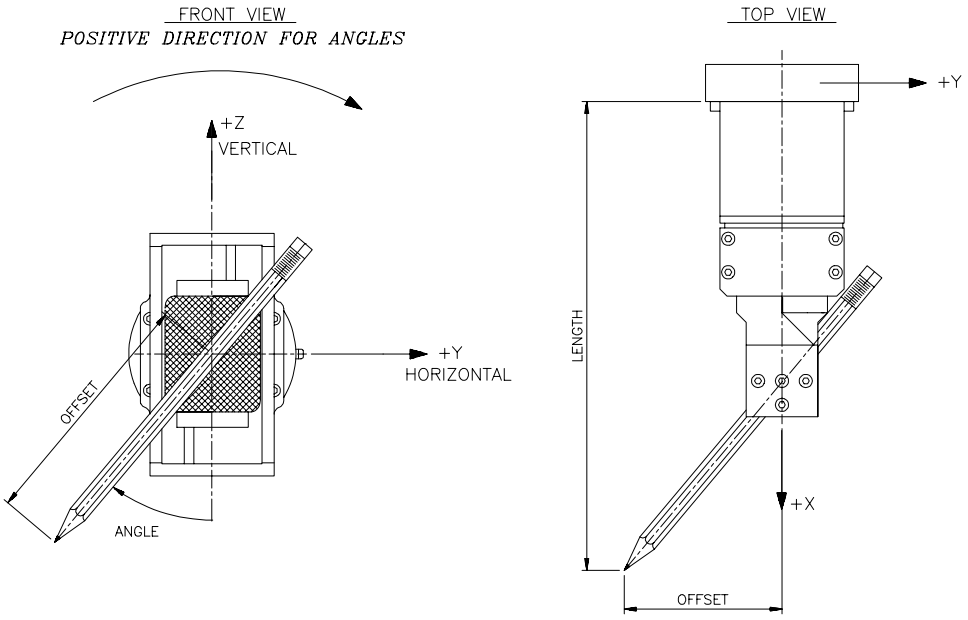
Where: *length* is the distance from flange to the tool center point (TCP); defined in microns.

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 vertical downward axis when link 4 is horizontal and roll is 0; 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).



The TOOL command sets the values of parameters 308, 309 and 310 (i.e., length, offset and angle, respectively). When the parameter values are defined by the user, default settings for TOOL can be loaded from a parameter backup file.

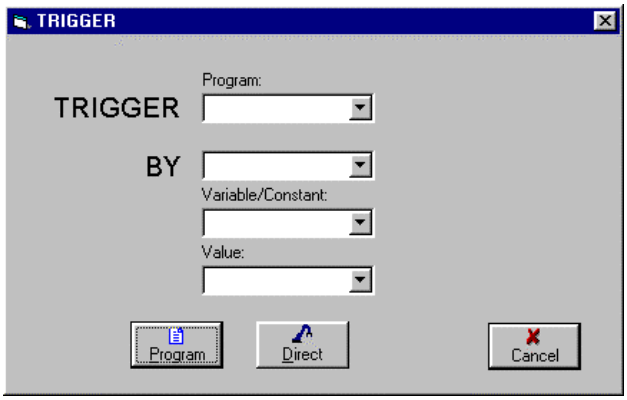
After the TOOL command is executed and parameters 308, 309 and 310 are updated, the values of tool parameters 311, 312 and 313 are also updated.

Example: ■ (*Define global variables*: 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
```

Notes: See also the TCP command.

TRIGGER



Format: TRIGGER *prog* BY {IN/OUT} *n* (0/1)

Where: IN is an input;
OUT is an output;
n is the I/O index: $1 \le n \le 16$ default
$1 \le n \le 48$ if I/O expansion card installed.

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 W BY OUT 8 — Program W is activated when output 8 changes its state.

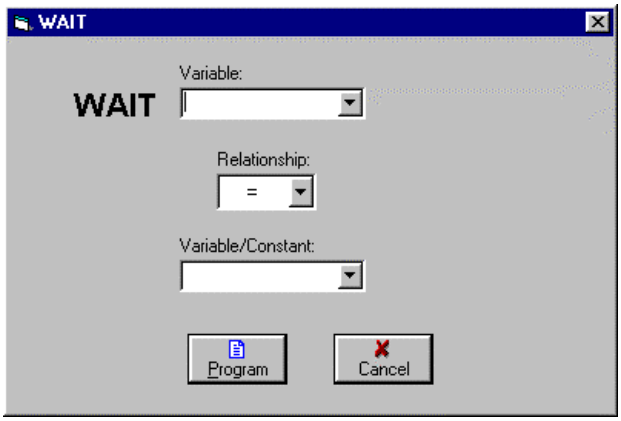
■

```
PROGRAM DRILL
********************
MOVE P28
SET OUT[3]=1
DELAY 500
SET OUT[3]=0
MOVE P27
TRIGGER DRILL BY IN 15 1
END

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

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

WAIT



Format: WAIT *var*1 *oper var*2

Where: *var*1 is a variable;
*var*2 is a variable or a constant;
oper can be: <, >, =, <=, >=, <>

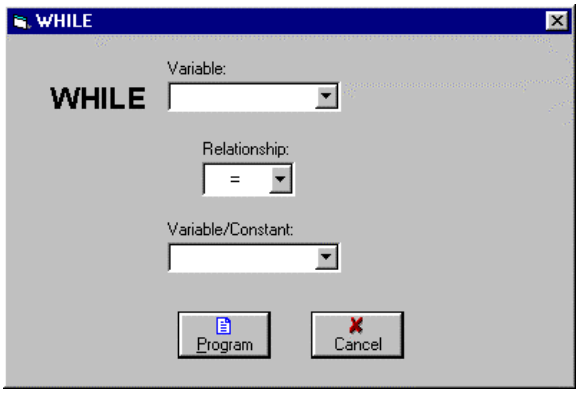
Description: Program execution is suspended until the specified condition is true.

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.

WHILE



Format: WHILE *var1 cond var2*

Where: *var*1 is a variable;
*var*2 is a variable or a constant;
o*per* can be: <, >, =, <=, >=, <>

Description: Executes a block of code as long as the specified condition is true.

The last line of the block must be the ENDWHILE command.

Example: ■

```
WHILE A < 50
    MOVED POS[A]
    SET A=A+1
ENDWHILE
```

Note: See also the ENDWHILE and FOR commands.

ZTOOL

This is a Direct command which does not open a dialog box.

Format: ZTOOL

Description: Defines the orientation of the tool frame relative to the default tool frame.

Before executing ZTOOL, move the robot until the following conditions are met:

- The desired Z tool direction is along world Z, pointing down.
- The desired X tool direction is along world Y in negative direction
- The desired Y tool direction is along world X direction, in negative direction.

The orientation is saved in 9 parameters, from 341 to 349. Parameters values contains the coordinates of the unity vectors of the tool frame relative to the default tool frame, normalized to 1,000,000. Consequently Default values are:
par[341..349]=[(1000000,0,0),(0,1000000,0),(0,0,1000000)]

6

Variables

Types of 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 predefefined **system** variables.

User variables are either **global** (recognized by all programs in a project), or **local** (i.e., **program** variables, recognized only by the program for which it is defined.)

A variable may be singular or an array. Names of array variables also include an index (a number within square brackets) which defines the number of variables in the array.

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

Local variables, like global variables, maintain only one value, even when multiple instances of the program are being executed.

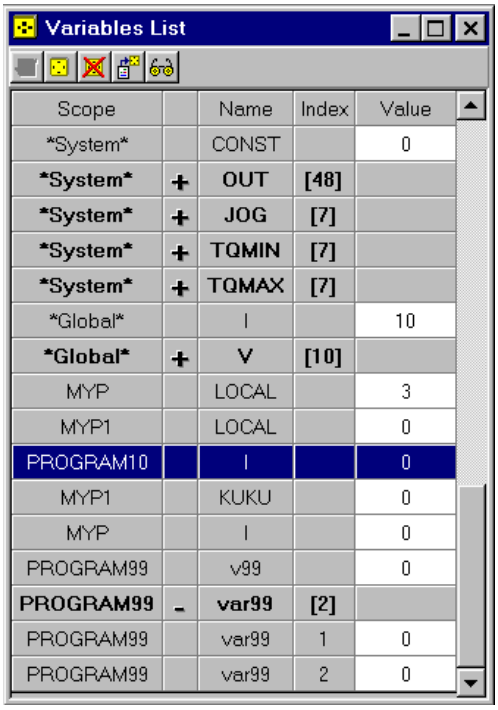
System variables are described in detail later in this chapter.

Using Variables

The Variables List dialog box displays and allows you to define and manipulate variables.

To access this dialog box, do either of the following:

- Select Variables in the Project tree.
- Select **View | Variables**.



Scope		Name	Index	Value
System		CONST		0
System	+	**OUT**	**[48]**	
System	+	**JOG**	**[7]**	
System	+	**TQMIN**	**[7]**	
System	+	**TQMAX**	**[7]**	
Global		I		10
Global	+	**V**	**[10]**	
MYP		LOCAL		3
MYP1		LOCAL		0
PROGRAM10		I		0
MYP1		KUKU		0
MYP		I		0
PROGRAM99		v99		0
PROGRAM99	-	**var99**	**[2]**	
PROGRAM99		var99	1	0
PROGRAM99		var99	2	0

The Variables List displays the following information:

Scope

Defines the type of variable:
- **System** for system defined variables.
- **Global** for global variables
- **Program name** for local variables.

If the variable is an arrray, a + sign will appear. Click on the + sign to expand and view the list of elements in the variable array.
Click on the – sign to contract the list.

Name

Name of the variable.

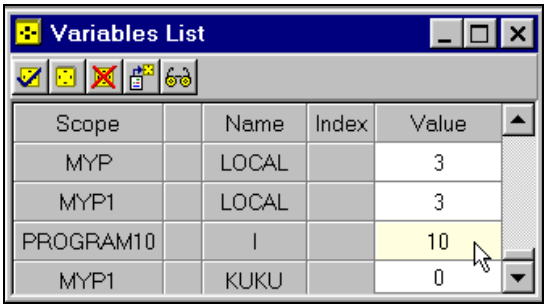
Index

This number identifies a specific element within an array variable.

Value

Value of the variable the last time it was uploaded or downloaded.

This field turns yellow when the user enters a new value. The field reverts to white after Apply is selected.



Since variable values can change rapidly within the controller, the value displayed in this window is updated only during uploading and refreshing.

To track changing variable values online, you must define a variable as a "watched variable" by means of the **Add to Watch** function. The **Watch Variables** window is then used for online tracking.

The short-cut buttons in this window provide the following functions:

Apply

When the user changes a variable value (in a field in the Value column) during online system operation, Apply causes the new settings to take effect within the controller.

New

Opens the Define Variable dialog box for defining a new user-defined global variable. (Local variables are defined within the program to which they are dedicated.)

Delete

Deletes the global variable currently selected in the Variable List. (Local variables are deleted within the program to which they are dedicated.)
If the selected variable is an array element, you will be prompted whether or not to delete all variables in the array. You cannot delete one element of a variable array.

Reload from Controller

Loads (to PC memory) variables values from the controller, and refreshes (resets) the values displayed in the dialog box. Available only when the system is operating online.

Add to Watch

Places the selected variable on the **Watch Variables** list, which are displayed in the Watch Variables window.
Up to 20 variables can be watched at the same time.

Variables may also be accessed and manipulated by ACL commands, as shown in the following examples:

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.

Defining Variables

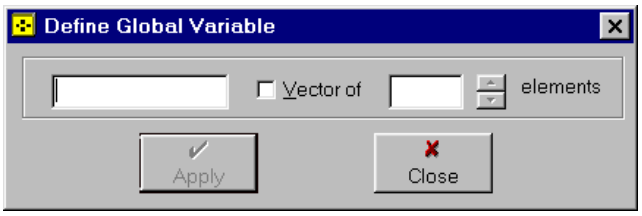
Variables must be defined before they can be used in a program.

Global Variables

From the Variable List or Project tree, select **New**.

The Define Variable dialog box opens.

Complete the fields to define a global variable and either a single or vector (array) variable.



Apply	Creates new variable according to name and definitions currently displayed in the dialog box..
Close	Closes the Define Variable dialog box.

Unlike most other dialog boxes, this dialog box remains open after you select Apply. This allows you to define a series of variables without having to reopen the dialog box repeatedly.

Local Variables

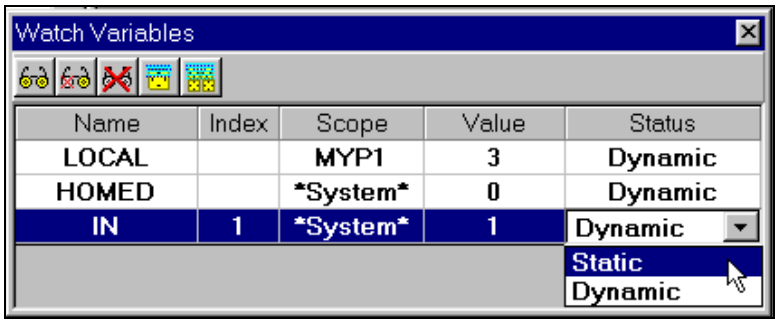
From the command list, select #LOCAL to define a local variable.

In the Define Variable dialog box complete the fields to define a local variable and either a single or vector (array) variable.

Refer to the command #LOCAL in Chapter 5.

Watching Variables

Since variable values can change so rapidly within the controller, the controller will only report changes in the values of variables which have been defined as Watched Variables.



The Watched Variables List displays the following information:

Name	Name of the watched variable.
Index	The number of a specific variable within a vector (array) variable.
Scope	The type of variable: • **System** for system defined variables. • **Global** for global variables. • Program name for local variables.
Value	Current value of the variable.
Status	Allows you to freeze the displayed value of a watched variable. • **Dynamic**: (default setting) Continuously refreshes the displayed value. • **Static**: Freezes the displayed value of the variable.

Up to 40 variables can be on the Watch Variables list. When you add the 41st variable to the list, the first variable on the list is removed.

The Watch Variables dialog box has the following shortcuts:

	Add	Displays the List Variables dialog box, which allows to add another variable to the watched list.
	Delete	Deletes a selected watched variable.
	Delete All	Deletes all defined watched variables.
	Refresh	Updates the value of a selected variable when the Refresh Status is set to Static.
	Refresh All	Updates the values of all variables

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.

System variables can be used in the same manner as user variables. However, system variables cannot be deleted.

ACL contains 22 system variables:

CONST	HOMED	LSCCW[n]	TIME
CPOS[n]	IN[n]	LSCW[n]	TQ[n]
ENC[n]	INDEX[n]	LTIME[n]	TQMAX[n]
ERRLI[n]	JOG[n]	MFLAG[n]	TQMIN[n]
ERROR[n]	JTARG[n]	OUT[n]	XTARG[n]
ERRPR[n]		POSER[n]	

The index [*n*] indicates a variable vector.

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 some system variables 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, ENC, LSCW, LSCCW, CPOS, POWER, JTARG, XTARG, TQ, INDEX, TIME, CONST, LTIME, MFLAG, and HOMED and JOG are read-only variables.

ERROR, ERRPR, ERRLI, OUT, TQMAX and TQMIN are read-write variables.

IN[*n*]

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

See also the ENABLE, DISABLE and FORCE commands, and the View | Status Bars | Inputs option.

Example

Purpose	Program Command
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.

See also the View | Status Bars | Encoders option.

Example

Purpose	Program Command	Notes
To assign the encoder value to a variable.	SET X=ENC[5]	The value of encoder 5 is written to X.

LSCW[*n*] and LSCCW[*n*]

The value of the variables LSCW[*n*] and LSCCW[*n*] indicates the status of the clockwise and counter-clockwise limit switches, respectively, for the specified axis at its current position.

In order to obtain a correct value when manually moving the axis from one limit to the opposite one, you must press the controller's Error Reset button.

These are read-only variables.

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 LSCW[4] DELAY 100 GOTO 1	Depending on the actual wiring connections, the value of LSCW will change to either 1 or 0 when the clockwise limit switch is detected

CPOS[*n*]

This variable contains the current command 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.

POSER[*n*]

This variable contains the position error; that is, the difference between the exact command position, as calculated by the driver, and the actual position, as measured by the encoder. 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.

JTARG[*n*] and XTARG[*n*]

JTARG[*n*] and XTARG[*n*] contain the Joint / World (XYZ) coordinates of the final target position for the movement command currently being executed.

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.

TQ[*n*]

This variable contains a value proportional to the torque output of the motor.

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.

The value of TQ[*n*] is in the range: ±5000, where 5000 represents the maximum peak torque for the specified motor.

INDEX[*n*]

This variable contains the index number of the next position to be passed during the execution of a MOVES movement.

This is a read-only variable.

For the robot ax, *n* =1; for the peripheral axis, *n*=2.

Example

Purpose	Program Command	Notes
Allows changes to speed during different sections of MOVES movements	MOVES M 1 100 FOR K = 2 TO 100 WAIT INDEX[1]>=K GSPEED SP[K] ENDFOR	Each section of the MOVES movement is monitored. When section K is reached,, the appropriate global speed is set.

TIME

This variable contains the current value of the real-time clock. When the controller is turned on or reset, the clock is initialized to 0. Every 10 milliseconds the clock value is incremented by 1.

TIME is considered a read-only variable.

Example

Purpose	Program Command	Notes
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

LTIME[*n*]

The value 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. The actual arrival time may be slightly later than the value of LTIME[*n*]. LTIME[*n*] is considered read-only variables.

For robot, *n* =1; for peripheral axis, *n*=2.

This variable is 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.

Example

Purpose	Program Command
To synchronize the arrival of robot and peripheral axis at their respective destinations. (POSR5 is a robot position and POSP3 is a peripheral axis position.)	PROGRAM SYNCH| ********** MOVE POSR5 SET V=LTIME[1]-TIME MOVE POSP3 V

CONST

This value of this variable indicates whether or not an axis group or an axis is in CON.

CONST is considered a read-only variable.

Whenever a CONcommand is executed, the 8 least significant bits of the variable CONST are switched on, according to the following:

- Bit 1 is ON If robot is in CON state
- Bit 2 is ON If peripheral axis exists and is in CON state

Use the AND operator of the SET command to mask the unwanted bit(s) of the CONST variable.

(The CONST variable functions in the same manner as the MFLAG variable; see following.)

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 8 least significant bits of the variable MFLAG are switched on, according to the following:

- Bit 1 is ON if robot is in motion
- Bit 2 is ON if peripheral axis is in motion

Example

Assuming the controller is configured with six robot axes and one peripheral axis (axis 7), the value of MFLAG will indicate movement of the axes as shown in the following chart:

Bit Value	1	2
Axis Group	Robot	Peripheral
Axis	1,2,3,4,5,6	7

Purpose	Program Command	Display	Notes
Movement status of the axes	PRINTLN MFLAG	1	All robot axes are in motion.
		3	Peripheral axis is in motion.

Use the AND operator of the SET command to mask the unwanted bit(s) of the MFLAG variable; for example: SET M=MFLAG AND 1 will give the movement status of the robot and mask all other axis groups.

HOMED

This variable indicates whether or not an axis has been homed.

HOMED is considered a read-only variable.

Whenever an axis group or an axis has been homed, the 32 bits of the binary representation of HOMED are switched on, according to the following:

1st (least significant) bit is set to 1 if axis 1 has been homed
2nd bit is set to 1 if axis 2 has been homed.
3rd bit is set to 1 if axis 3 has been homed.
4th bit is set to 1 if axis 4 has been homed.
5th bit is set to 1 if axis 5 has been homed.
6th bit is set to 1 if axis 6 has been homed.
7th bit is set to 1 if axis 7 has been homed.
Bits 8 through 32 are always set to 0.

ERROR, ERRPR and ERRLI

These are array variables. When a system error occurs during run time, these 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) 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.

These variables are displayed in the Variables List in last-to-first order, so that the most recent errors appear at top of the list.

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 PROGRAM ERROR ********** SET ERROR=0 SET ERRPR=0 SET ERRLI=0 * wait for error to occur WAIT ERROR <>0 PRINTLN "ERROR NO. " ERROR PRINTLN "TASK ID " ERRPG PRINTLN "AT LINE " ERRLI	 53 7 144	Simultaneously run programs ERROR and MOVE. Error ID number Task number 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.

Example

Purpose	Program Command	Notes
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]	

JOG[*n*]

Warning: This variable is for maintenance purposes only. Do not manipulate this variable unless you are authorized to do so!

The value of this variable defines a constant speed for movement of an axis.

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

The value of JOG is in units of (encoder counts x 128) / controller clock tick.

Example

Purpose	Notes
SET JOG[3]=12800 DELAY 100 SET JOG[0]=0	This will move axis 3 at a relatively slow speed of 100 encoder counts per controller clock tick.

TQMAX[*n*] and TQMIN[*n*]

TQMIN and TQMAX contain the minimum and maximum torque setting for the specified axis.

TQMIN and TQMAX are two clamping values which limit the working range of the servo controller. If the axis reaches either of these limits during movement, the axis' position error will increase, but the torque will remain within the defined range. If the position error reaches the maximum allowed, as defined by the impact parameters, an impact error condition will result.

TQMIN and TQMAX are automatically reset to -1000 and +1000, respectively, when variable values are reloaded from the controller.

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

The torque limits are written to the axis controller only when the value of TQMAX is set. Thus, to change only TQMIN, you must rewrite the value of TQMAX.

The value of TQMIN and TQMAX is in the range: ±5000.

Example

Purpose	Program Command	Notes
To limit the torque during execution of a delicate task.	SET TQMAX[4]=950 SET TQMIN[4]=600	Monitoring variable TQ[4] during program execution showed a torque range of 700-850 during the delicate segment of the program. Setting the torque limits in the range 600-950 protects against impact conditions.

7

Maintenance

The Maintenance menu provides direct access to controller functions that are not part of ACL projects.

The options in the Maintenance menu are intended for qualified technicians.

Robot Home

The HOME command is used to define the robot home position.

Homing is required only at first installation and after maintenance.

To home the robot, do the following:

1. Before executing the HOME command, manually bring each of the three links on the robot arm to the positions marked on them.
2. Then select select **Robot | Home**.

This will record the coordinates of the robot's current location to system defined position $HOME.

Do not attempt to alter the coordinates of $HOME through the Teach Position dialog box.

Refer to the section "System Positions" in Chapter 3.

To home the robot from the teach pendant, press the keys on the TP:
[RUN] [0] [0] [Enter]

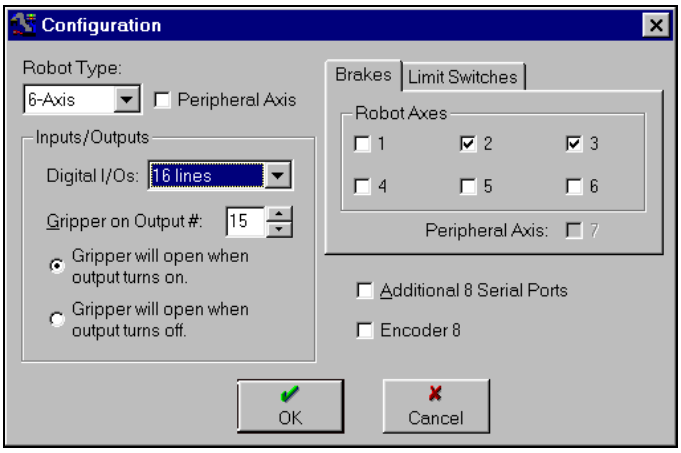
Robot-Controller Configuration

Configuration settings are automatically read from the controller when the system is operating online.

Do not change the configuration settings unless you have changed hardware connections.

When hardware connections are altered, the data in this dialog box must be changed accordingly.

1. Select **Robot | Configuration**.
   The Configuration dialog box opens.



2. Make sure the default configuration settings are selected:
   - Robot: 6-axis
   - Peripheral Axis: not selected
   - Digital I/Os: 16 inputs, 16 outputs
   - Additional 8 Serial Ports: not selected
   - Encoder 8: not selected
   - Brakes: axes 2 and 3 selected; axes 1, 4, 5 and 6 not selected
   - Limit Switches: no axes selected
3. Define the number of the output to be used for controlling gripper operation. Also define the input state which will cause the gripper to close.

Terminal

Terminal opens an ACL-DOS window. It is intended for users familiar with ACL programming in the DOS environment.

Parameters

About Parameters

Parameters are reserved memory locations which are used to set the values of physical constants needed to adapt the controller to a particular robotic system. Parameters are referred by their number (1 to 2024). For example:

```
LET PAR 294 8000        Sets value of parameter 294 to 8000.
```

Many of the controller functions depend on the setting of the system parameters. System parameters determine operations and conditions such as work envelope, axis protection, speed limits, gripper operation, teach pendant and manual operation, kinematic calculations, and more.

Warnings

- **Only skilled operators should attempt to manipulate parameters**.
- Backup current system parameters before you change parameter values.
- Activate COFF before you change parameter values.
- Never change parameter values while the robot is in motion.
- Never change parameters values while programs are running.
- Be sure impact protection parameters are properly set. These parameters monitor the 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.

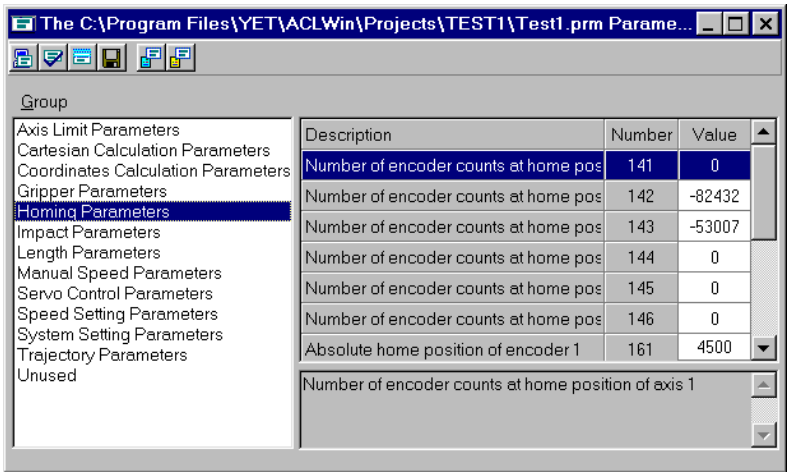
Manipulating Parameters

The Parameters List dialog box displays and enables manipulation of system parameter values.

To access the parameters dialog box, select **Maintenance** | **Parameters**.

Then select either of the following options:

- **Load from Controller**. Loads parameter values from the controller to the list of parameters.
- **Open File**: Loads parameter values from a parameter file. This option opens a file dialog box that lets you select a parameter file. Parameter files have the extension **prn**.



The Parameters List contains the following information:

Group	Parameter category.
Description	Start of parameter description. The complete description appears in the text box at the bottom of the dialog box.
Number	Parameter identification number.
Value	Shows current parameter values in PC memory. When **online**, select Refresh to upload and display values from controller.

The short-cut buttons in this dialog box provide the following functions:

Load Parameters from File	Loads parameter settings from current project's parameter file to PC memory) Useful for resetting parameters which were changed. Available only when the system is operating offline.
Apply	Sends new parameter value to controller (but does not activate). Allows you to send a set of new values to the controller, which can then be activated simultaneously. Available only when online.
Refresh	Loads (to PC memory) parameter settings from the controller, and refreshes (resets) the values displayed in the dialog box. Available only when the system is operating online.
Save	Saves the parameter values to a user defined file.
Activate Parameter Set	Resets the control process in the controller according to all new parameter values which were sent to the controller (INIT CONTROL).

Load Servo Parameter	Resets the values of axis driver parameters according to the new parameter values which were sent to the controller. After you select Load Servo, you are prompted to define the axis for which the parameters are being loaded.

Parameters may also be accessed and manipulated by the following ACL commands:

LET PAR *n*=*var*	Changes the value of parameter *n* to *var* (either a constant or a variable).
SET var1=PAR *n*	Assigns the value of parameters to a variable.

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 239 defines the accleration at the start of a manual 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 200+*axis*.

Parameters 243, 263 and 283, for example, are parameters for axis 3; parameters 262, 263 and 264 define the position error constants for axes 2, 3, and 4, respectively.

Parameters values are expressed in units such as encoder counts, controller clock ticks, linear measurements, and so on.

The parameters supplied with the robot are appropriate for most robotic applications.

Do not change parameters unless necessary.

Read the documentation carefully before making any changes to parameters.

The following two tables describe all the parameters for Controller-BRC.

- 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.

Parameter Table 1		
System Setting Parameters		
I/Os for Remote (External) Control	14	Defines controller input # for external Servo ON switch.
	15	Defines controller input # for external Run/Hold switch.
	16	Defines controller input # for external Start switch.
	114	Defines controller input # for external Error Reset switch.
	115	Defines controller output # for external Error indicator.
	116	Defines controller output # for external Remote indicator.
	117	Defines controller output # for external Program Running indicator.
	118	Defines controller output # for external Hold indicator.
	119	Defines controller output # for external Servo indicator.
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.
Impact Parameters		
	240	Defines axis group which responds when impact detected.
Manual Mode	280 + *axis*	Maximum torque allowed while moving the axes manually (teach pendant).
	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.
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 acceleration.
	535	Maximum roll acceleration.
	536	Maximum linear speed.
	537	Maximum pitch speed .
	538	Maximum roll speed.

Parameter Table 1		
Manual Speed Parameters		
	220 + *axis*	Speed setting for manual operation.
	239	Acceleration at start of a manual movement.
	500 + *axis*	Deceleration of movement after TP key is released.
	238	Deceleration of Cartesian movement after TP key is released.
	294	Maximum speed for manual Cartesian movement.
Homing Parameters		
	140 + *axis*	Encoder reading at home position.
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	Number of encoder counts for +90° of axis 1.
	34	Number of encoder counts for +90° of axis 2.
	35	Number of encoder counts for +90° of axis 3.
	36	Number of encoder counts for +90° of axis 4.
	37	Number of encoder counts for +90° of axis 5.
	38	Number of encoder counts for +90° of axis 6.
Horizontal Reference Position	1	Encoder 1 reading at horizontal reference position
	2	Encoder 2 reading at horizontal reference position.
	3	Encoder 3 reading at horizontal reference position.
	4	Encoder 4 reading at horizontal reference position.
	5	Encoder 5 reading at horizontal reference position.
	6	Encoder 6 reading at horizontal reference position.
	260	Compensation of roll angle to maintain orientation of tool. For 5-axis robot only.
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.

Parameter Table 1		
	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.
	311	X coordinate of the TCP in the tool frame.
	312	Y coordinate of the TCP in the tool frame.
	313	Z coordinate of the TCP in the tool frame.
	314	Angle between upper arm axis of symmetry and the line passing through axes 3 and 4. (See Encoder Origin diagram.)
	237	Cartesian limit.
Trajectory Parameters		
	520 + *axis*	Maximum acceleration/deceleration.
	580 + *axis*	Deceleration smoothness.
	199	Rate of deceleration when Hold switch is pressed.
	260 + *axis*	Maximum position error at end of MOVED, MOVELD or MOVECD.
	800 + *n*	Bandwidth of the smoothing filter for trajectory. (*n*=1 for group A; *n*=2 for group B; *n*=axis number for individual axis in group C or gripper.
Gripper Parameters		
	76	Duration of gripper closing and opening.
	274	Controller output # for pneumatic gripper.
Servo Control		
	1010 + *axis*	Motor Winding Inductance.
	1020 + *axis*	Motor Torque Constant.
	1050 + *axis*	Maximum Motor Rated Current.
	1060 + *axis*	Phase Offset.
	1070 + *axis*	Number of Pole Pairs.

Parameter Table 1		
	1080 + *axis*	Motor Winding Resistance.
	1090 + *axis*	NKd. Maximum gain increase with speed.
	1100 + *axis*	DC Bus Voltage Compensation Coefficient.
	1114	Axis 4 Delay between COFF and brake activation
	1120 + *axis*	Delay between COFF off and axis disabled.
	1134	Axis 4 only Delay between CON and brake release.
	1140 + *axis*	dIdT Rate.
	1150 + *axis*	Proportional Feedback Gain.
	1160 + *axis*	Differential Feedback Gain.
	1170 + *axis*	Integral Feedback Gain.
	1180 + *axis*	Gain changing coefficient with position error.
	1190 + *axis*	Inductance Compensation Coefficient.
	1200 + *axis*	Additional Proportional Feedback Gain.
	1210 + *axis*	Maximum Thermic Load.
	1220 + *axis*	Maximum Proportional Feedback. In units of rpm.
	1230 + *axis*	Acceleration Feed Forward.
	1240 + *axis*	Minimum position error for using speed integral
	1250 + *axis*	PWM Threshold Value.
	1260 + *axis*	Maximum Current.
	1280 + *axis*	Maximum increase of variable gain.
	1310 + *axis*	Low Pass Filter on Voltage.
	1320 + *axis*	Smoothing of Command.
	1330 + *axis*	Smoothing of Feedback Torque.
	1340 + *axis*	Smoothing of Output Voltage.
	1350 + *axis*	Smoothing of Reactive Voltage.
	1360 + *axis*	Smoothing of the dIdT.
	1370 + *axis*	Reduction of Current Limit by dIdT.
	1380 + *axis*	Recording Option.
	1400 + *axis*	Recording Rate
	1440 + *axis*	Variable gain filter

Parameter Table 1		
	1450 + axis	PWM offset.
	1460 + axis	Minimum Speed at which variable gain begins to increase.
	1480 + *axis*	Thermic Overload Bleeding.
	1490 + *axis*	Deceleration Rate for Stopping.

Parameter Table 2	
Parameter	**Description**
1 2 3 4 5 6	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. Reading of encoder 6 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. Value usually = 0 (encoder origin)
14	Servo On Switch Defines the controller input to which a remote Servo On switch is connected. PAR 16 may have a value of 1–16. If PAR 16=0: switch not installed; not defined.
15	Run/Hold Program Switch. Defines the controller input to which a remoteRun/Hold switch 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 Switch. Defines the controller input to which a remote Start program switch is connected. PAR 16 may have a value of 1–16. If PAR 16=0: switch not installed; not defined.
33 34 35 36 37 38	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. Number of encoder counts for +90° of axis 6.
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. OPEN/CLOSE commands will activate output which controls gripper, then delay, per PAR 76, before next ACL command is executed.
100 + *axis*	Upper Limit of axis motion, in encoder counts, per axis.

Parameter Table 2	
Parameter	**Description**
114	Error Reset Switch. Defines the controller input to which a remote Error Reset switch is connected. PAR 114 may have a value of 1–16. If PAR 114=0: switch not installed; not defined.
115	Error Indicator. Defines the controller output to which an external Error indicator is connected. PAR 115 may have a value of 1–16. If PAR 115=0: indicator not installed; not defined.
116	REMOTE Indicator. Defines the controller output to which an external REMOTE mode indicator is connected. PAR 116 may have a value of 1–16. If PAR 116=0: indicator not installed; not defined.
117	Program Running Indicator. Defines the controller output to which an external Program Running indicator is connected. PAR 117 may have a value of 1–16. If PAR 117=0: indicator not installed; not defined.
118	Hold Indicator. Defines the controller output to which an external Hold Program indicator is connected. PAR 118 may have a value of 1–16. If PAR 118=0: indicator not installed; not defined.
119	Servo Indicator. Defines the controller output to which an external Servo On indicator is connected. PAR 119 may have a value of 1–16. If PAR 119=0: indicator not installed; not defined.
120 + *axis*	Lower Limit of axis motion, in encoder counts, per axis.
140 + *axis*	Encoder reading at home position.
180 + *axis*	Maximum Speed Setting. For robot axes: in (degrees/second). For non-robot axes: in (encoder counts/10 milliseconds).
199	Rate of Deceleration of axis motion when the Hold switch is pressed. Defined as a percentage of maximum speed, as defined by PAR 180+*axis*: 1=slow; 100=immediate. Typical value: 4–5.
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*.
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.

Parameter Table 2	
Parameter	**Description**
238	Manual Cartesian Movement Deceleration. The rate of deceleration after a Cartesian manual movement TP 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 accelerationat 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).
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). For 5-axis robot only. 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.)
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 software 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.
280 + *axis*	Manual Movement Torque Limit. The maximum torque allowed while moving the axes manually (by means of TP or software). DAC value: 0–5000.
294	Manual Cartesian Movement maximum speed. A percentage of maximum linear speed, as defined by PAR 536.
301	X coordinate of the rotation axis of arm link 2 when the robot is in the home position. Defined in microns.

Parameter Table 2	
Parameter	**Description**
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. Defined in microns. Calcuated from PAR 307 and PAR 308.
307	Distance from pitch axis to flange; defined in microns. 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.
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 and roll is 0; defined in thousandths of a degree.
311	X coordinate of the TCP in the tool frame.
312	Y coordinate of the TCP in the tool frame.
313	Z coordinate of the TCP in the tool frame.
314	Angle between upper arm axis of symmetry and the line passing through axes 3 and 4; defined in thousandths of a degree. (See Encoder Origin diagram.)
500 + *axis*	Manual Movement Deceleration. The rate of deceleration after the manual movement TP 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. Clock tick = 10 ms.
533	Maximum Linear Acceleration in microns/(hundreths of a second)^2.
534	Maximum Pitch Acceleration, in millidegrees/(hundreths of a second)^2.
535	Maximum Roll Acceleration, in millidegrees/(hundreths of a second)^2.
536	Maximum Linear Speed, in (microns/second).
537	Maximum Pitch Speed, in (millidegrees/second).
538	Maximum Roll Speed, in (millidegrees/second).

Parameter Table 2	
Parameter	**Description**
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.
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).
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 value: 1000–2000 encoder counts.
1010 + *axis*	Motor Winding Inductance. In units of 0.02 mH
1020 + *axis*	Motor Torque Constant. In units of (mN x m / A)
1050 + *axis*	Maximum Motor Rated Current. In units of mA.
1060 + *axis*	Phase Offset.
1070 + *axis*	Number of Pole Pairs.
1080 + *axis*	Motor Winding Resistance. In units of mΩ.
1090 + *axis*	NKd. Maximum gain increase with speed. Defined as percentage of target speed.
1100 + *axis*	DC Bus Voltage Compensation Coefficient. In units of (0.01 V / digit).
1114	Axis 4 only. Delay between COFF and brake activation. In units of 10 ms.
1120 + *axis*	Delay between COFF off and axis disabled. In units of 10 ms.
1134	Axis 4 only. Delay between CON and brake release. In units of 10 ms.
1140 + *axis*	dIdT Rate. Defined as percentage of current limit.
1150 + *axis*	Proportional Feedback Gain.
1160 + *axis*	Differential Feedback Gain.
1170 + *axis*	Integral Feedback Gain.

Parameter Table 2	
Parameter	**Description**
1180 + *axis*	Gain changing coefficient with position error. In units of (%/mrad)
1190 + *axis*	Inductance Compensation Coefficient. In microseconds.
1200 + *axis*	Additional Proportional Feedback Gain.
1210 + *axis*	Maximum Thermic Load.
1220 + *axis*	Maximum Proportional Feedback. In units of rpm.
1230 + *axis*	Acceleration Feed Forward.
1240 + *axis*	Minimum position error for using speed integral. In units of (2^{20}/turn).
1250 + *axis*	PWM Threshold Value.
1260 + *axis*	Maximum Current. A percentage of the Rated Current as defined by PAR 1050 + *axis*.
1280 + *axis*	Maximum increase of variable gain. Defined as a percentage of nominal gain.
1310 + *axis*	Low Pass Filter on Voltage. In microseconds.
1320 + *axis*	Smoothing of Command. In microseconds.
1330 + *axis*	Smoothing of Feedback Torque. In microseconds.
1340 + *axis*	Smoothing of Output Voltage. In microseconds.
1350 + *axis*	Smoothing of Reactive Voltage. In microseconds.
1360 + *axis*	Smoothing of the dIdT. In microseconds.
1370 + *axis*	Reduction of Current Limit by dIdT. Defined as percentage of dIdT.
1380 + *axis*	Recording Option.
1400 + *axis*	Recording Rate. In units of 125 microseconds.
1440 + *axis*	Variable gain filter. In units of 0.1 ms.
1450 + axis	PWM offset. 100/1875 %
1460 + axis	Minimum Speed at which variable gain begins to increase. In units of 0.1 rpm.
1480 + *axis*	Thermic Overload Bleeding.
1490 + *axis*	Deceleration Rate for Stopping. In units of (61 turns / sec^2)

8

System Configuration

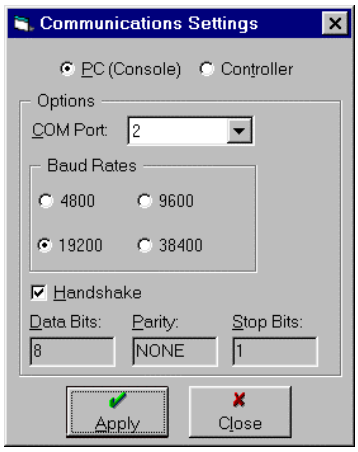
PC-Controller Communication

PC COM Settings

To establish communication between the PC and the controller, do the following:

1. Make sure you have made all the required hardware connections, as detailed in the controller's user manual.
2. Turn on the controller's Power switch.
3. Turn on your computer and activate Windows.
4. Click the ACL-Win icon on Windows desktop to load ACL-Win.
5. From ACL-Win, select **Communication | Settings**.
   The Communication Settings dialog box opens.
   - Make sure the PC COM Port setting is set to the COM port on the PC to which the controller is connected.

     By default, the COM port setting for **PC** is COM1. Change it if necessary, as shown in the figure below. Once changed, this setting will remain the default.



- Make sure the Baud Rate is set to 19200.
- Make sure Handshake is enabled.
- Do not change any other Communications settings.
- Select Apply and Close to accept the settings and close the dialog box.

Controller COM Settings

By default, the controller's CONSOLE COM port is connected to the PC COM port which is defined as the communication channel.

Do not manipulate the default settings unless you are authorized to do so.

Online Operation

In online operation, the PC communicates actively with the controller through ACL-Win and data in both the PC and the controller must remain identical to ensure proper operation.

Thus, when working online some data is automatically downloaded to the robot controller after it is entered through windows and dialog boxes, without needing to explicitly download entire data lists.

In addition, when working online, positions and variable values are read from the controller whenever you open or reload a list of positions or variables. To restore initial values, you must switch to offline mode and reload the data from the PC file.

Similarly, when operating online, parameter values are uploaded from the controller to the PC only when you open the list of parameters window.

Going Online

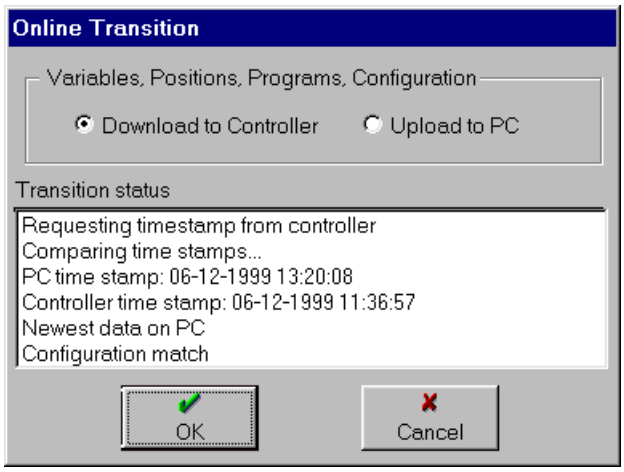
ACL-Win loads in offline mode, even when the controller is connected and turned on.

To ensure that data in PC and controller is identical when online operation begins, the system uploads data from the controller to the PC, or downloads data from the PC to the controller. Programd, positions and variables (but **not** parameters) are copied from the PC to the controller, or vice versa.

If you intend to work with project data which has been saved to files, be sure the project files are loaded before going online; select **File | Open Project**, and open the desired project.

To go online, select **Communication | Online** or click the online/offline toggle button:

The Online Transition dialog box opens. You are prompted to download or upload project data.



- **Download**. If you want to download project data from the PC to the controller from programming done offline or saved in files, select **Download** for all data options.
- **Upload.** If you want to upload data from the controller to the PC, do either of the following.
  - Select **Upload** for all data options.

  OR

  - **Cancel** the Online Transition dialog box.
  - Select **File | New Project**.
  - Select **Communication | Online**.

If no project data exists in PC memory when the system goes online, all data in the controller is *automatically* uploaded to the PC during the transition to online mode. No prompt appears.

If a connection cannot be established, check the PC – controller hardware connections and make sure the emergency button on the controller or teach pendant is not pressed.

Offline Operation

ACL-Win loads in offline mode, even when the controller is connected and turned on.

In offline operation ACL-Win and controller do not communicate, even if the PC and controller are connected. Offline operation allows you to program the robotic system without connecting the PC to the robot controller. Offline programming allows the robotic system to continue operating while new programs are being prepared.

You can, however, upload and download project data when the system is operating offline, **provided the controller is connected and turned on** (since the system goes online momentarily during offline uploading and downloading.) This is a useful and easy method, for example, for transferring data from one controller to another controller (upload and then download), or for backing up controller data.

The Upload and Download options in the Communication menu are available only when the system is operating offline.