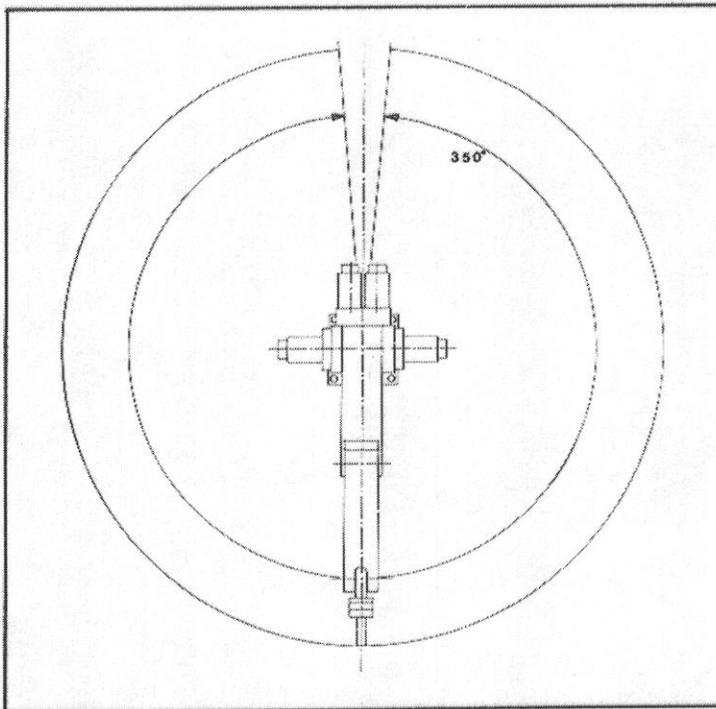


Workspace elevation - side view



Workspace elevation - top view

Specifications

Mechanical structure

Drive system: DC servo motors with optical encoders
 Transmission: Harmonic drives with preloaded drive chains
 Payload (maximum speed): 2.2 lbs (1.00 kg)
 Payload (reduced duty): 4.4 lbs (2.00 kg)
 Repeatability: +/- 0.005 in (0.13 mm)

Joint speed

Base: 60 degrees/second
 Shoulder: 60 degrees/second
 Elbow: 60 degrees/second
 Wrist: 180 degrees/second
 Tool: 180 degrees/second

Joint worst case resolution

Base: 0.0023 in
 Shoulder: 0.0023 in
 Elbow: 0.0014 in
 Wrist: 0.0025 in
 Tool: 0.0025 in

Robot controller

Control

Number of axes: Eight DC servos (5+3)
 Position detection: Digital optical encoders
 Point-to-point: Joint interpolated
 Straight line: At reduced speed
 Speed settings: 0 to 100 percent

Programming

Manual: Teach pendant
 Off-line: Resident RAPL language
 Editor: Resident RAPL editor
 Debugger: Resident RAPL debugger
 User memory size: 8K bytes

Auxiliary components

Communications: Dual RS232
 Digital inputs: Sixteen TTL
 (Optional buffering)
 Digital outputs: Sixteen TTL
 (Optional buffering)
 Expansion: Peripheral expansion bus

Other

Power requirements: 100-130 VAC, 50-60 Hz, 3A
 Mounting dimensions: 19 inches rack mountable

Options

Analog inputs
 Bubble memory
 Digital I/O buffering (AC/DC)
 Digital I/O expansion
 PC host interface
 Pneumatic gripper
 Servo gripper
 UNIX Host interface
 Video terminal

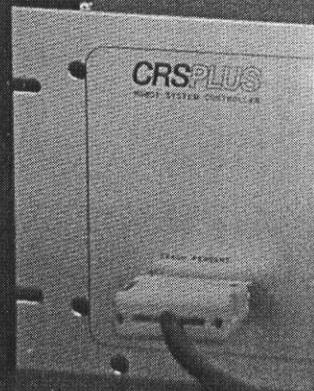
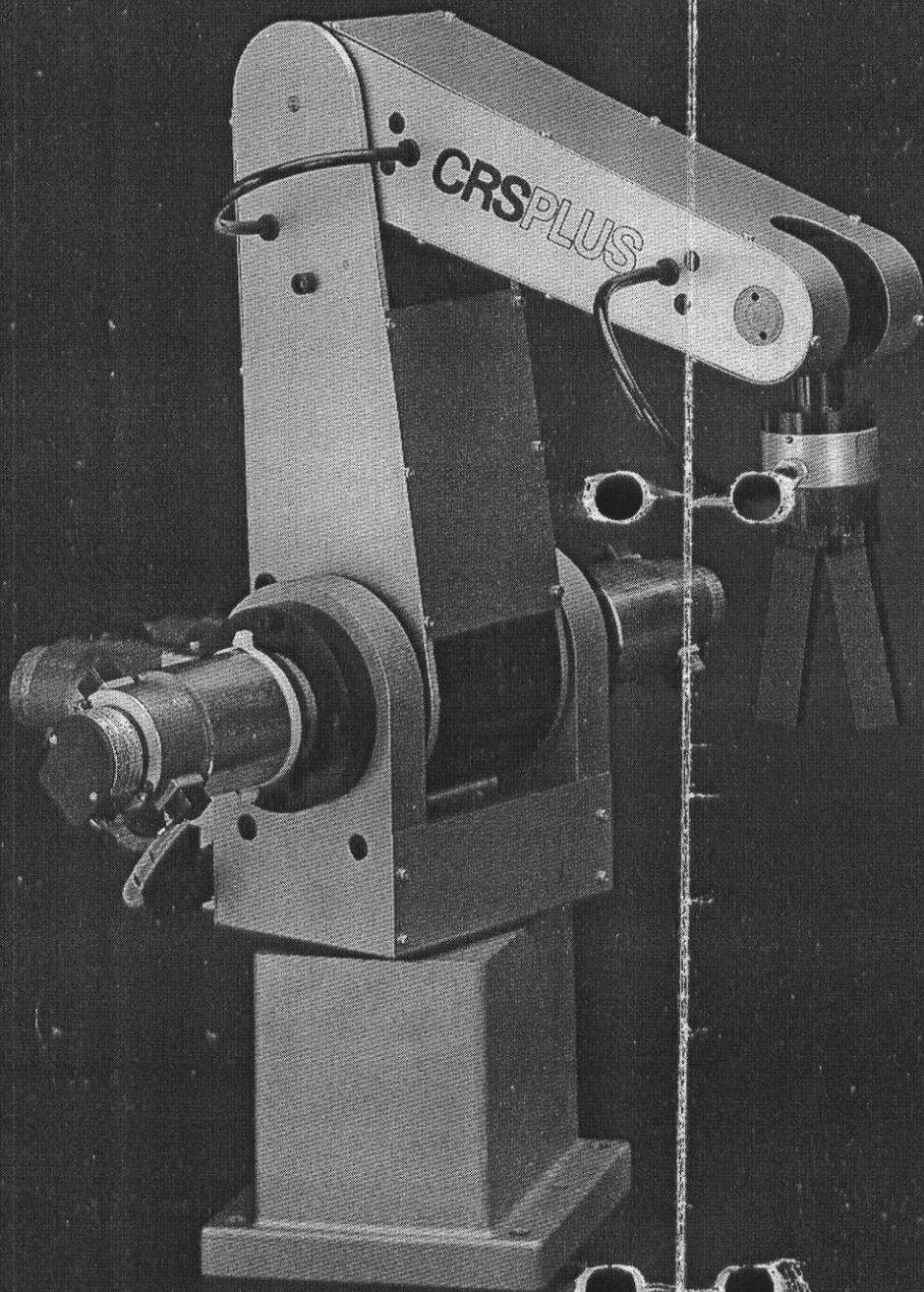
Specifications are subject to change without notice.

CRSPLUS
 INDUSTRIAL AUTOMATION

CRS PLUS INC.

835 Harrington Court
 P.O. Box 163, Station A
 Burlington, Ontario, Canada
 L7R 3Y2

Tel.: (416) 639-0086



SRS-M1A

**SMALL INDUSTRIAL ROBOT SYSTEM
RAPL PROGRAMMING MANUAL**

Manual Order Number: SRS-MAN/RAPL

REV.	REVISION HISTORY	DATE
-001	Original Issue	12/85
-002	RAPL V1.2	2/86
-003	Version 3.30	9/86
-004	Version 3.4.5	2/87
-005	Version 3.5.0	6/87
-006	Version 4.10	2/88

Additional copies of this manual or other CRS Plus literature may be obtained from:

CRS Plus Inc.
P.O. Box 163 Stn 'A'
Burlington, Ontario
L7N 3P3
CANADA

Phone: (416) 639-0086

The information in this document is subject to change without notice.

CRS Plus Inc makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. CRS Plus Inc assumes no responsibility for any errors that may appear in this document. CRS Plus Inc makes no commitment to update nor to keep current the information contained in this document.

CRS Plus Inc assumes no responsibility for the use of any circuitry other than circuitry embodied in a CRS product.

CRS Plus Inc software products shall remain the property of CRS Plus Inc.

PREFACE

This manual provides general programming information for the CRS Plus RAPL programming language. Additional information is available in the following documents:

- * SRS-M1A Small Industrial Robot System - Tutorial Manual
- * SRS-M1A Small Industrial Robot System - Technical Manual
- * SRS-M1A Small Industrial Robot System - IBM-PC Host Interface

CONTENTS

CHAPTER 1 - OVERVIEW	
1-1	RAPL DESCRIPTION 1-1
1-2	NOTATIONAL CONVENTIONS USED IN MANUAL 1-2
1-3	COMMAND ACCESSIBILITY 1-3
1-4	RAPL COMMAND FORMAT 1-3
1-5	NAMING CONVENTIONS 1-4
1-6	TOKENIZED FORMAT 1-5
CHAPTER 2 - PROGRAMMING CONSIDERATIONS	
2-1	PROGRAMMING AIDS 2-1
2-2	HELP MODE 2-1
2-3	DEFINING THE WORKSPACE 2-1
2-5	TOOL COORDINATE PROGRAMMING 2-3
2-6	MANUAL CONTROL 2-4
CHAPTER 3 - SUMMARY OF RAPL COMMANDS	
3-1	INTRODUCTION 3-1
3-2	MOTION COMMANDS 3-1
3-3	INPUT/OUTPUT COMMANDS 3-3
3-4	GRIPPER COMMANDS 3-4
3-5	SYSTEM COMMANDS 3-5
3-6	ROBOT LOCATION ASSIGNMENT 3-6
3-7	PROGRAM FLOW FUNCTIONS 3-8
3-8	STRING FUNCTIONS 3-9
3-9	MATHEMATICAL FUNCTIONS 3-11
3-10	EXTRA AXIS COMMANDS 3-12
3-11	LINE EDITOR 3-13
CHAPTER 4 - RAPL COMMANDS	
4-1	INTRODUCTION TO COMMAND LIST 4-1
CHAPTER 5 - LINE EDITOR	
5-1	DESCRIPTION 5-1
5-2	EDIT COMMANDS 5-2
APPENDIX A - COMPENDIUM OF RAPL COMMANDS	
APPENDIX B - RAPL ERROR LIST	
APPENDIX C - TERMINAL CONTROL CODES	
APPENDIX D - AUTOSTART PROCEDURE	
D-1	INTRODUCTION D-1
D-2	EXAMPLE 1 D-1
D-3	EXAMPLE 2 D-3

TABLES

TABLE 2-1	Switch functions in MANUAL Mode.	2-4
TABLE 3-1	Motion Command Summary	3-2
TABLE 3-2	Input/Output Command Summary	3-3
TABLE 3-3	Gripper Command Summary	3-4
TABLE 3-4	System Command Summary	3-5
TABLE 3-5	Robot Location Assignment Command Summary	3-7
TABLE 3-6	Programming Flow Command Summary	3-8
TABLE 3-7	RAPL String Functions	3-10
TABLE 3-8	Mathematical Programming Command Summary	3-11
TABLE 3-9	Extra axis commands	3-12
TABLE 3-10	Editor Command Summary	3-13

FIGURES

FIGURE 2-1	Defining the workspace - Real World Co-ordinates	2-2
FIGURE 2-2	Defining the workspace - Joint Motions.	2-2
FIGURE 2-3	Orientation of the tool coordinate system.	2-3
FIGURE 4-1	Arm "READY" Position	4-104

1-1 RAPL DESCRIPTION

The SRS-M1A Small Industrial Robot System uses the CRS Plus proprietary language known as RAPL (Robotic Automation Programming Language). RAPL is an automation-oriented line-structured language designed to facilitate the design of applications of robot systems.

RAPL uses english-like commands to provide a user friendly interface for the operator. Features of the RAPL language include efficient coding structures to optimize memory usage, alternate command identifiers, and advanced mathematical expressions.

1-2 NOTATIONAL CONVENTIONS USED IN MANUAL

NOTATION	DESCRIPTION
BOLD PRINT	Characters shown must be entered in the order shown.
VAR_NAME	Refers to a variable name.
PRG_NAME	Refers to a program name.
LOC_NAME	Refers to a Cartesian Point location name.
#LOC_NAME	Refers to a Precision Point name.
STR_NUM	Refers to the string number (1 - 4)
CHAR_INDEX	Refers to the character index within a string
[]	Square brackets indicate optional arguments or parameters.
[,...]	The preceding argument may be repeated, but each repetition must be separated by a comma.
< >	Triangular brackets indicate required arguments or parameters.
	Vertical bar separates options within [] or < > brackets.
,	Comma separates arguments.
+	High level true (input commands)
-	Low level true (input commands)
<cr>	Indicates a carriage return.
&	Indicates that a string number follows
^	Indicates an implied location name
<ABORT>	Indicates pressing <Ctrl-C>, <Ctrl-X> or the ABORT push-button on the Teach Pendant.

EXAMPLE: APPRO command

[line#] **APPRO** <LOC_NAME>, <distance>[, <S>]<cr>

ITEM	DESCRIPTION
[line#]	optional (line is needed for program entry only)
APPRO	required parameter (command name)
<LOC_NAME>	required location name
,	comma must separate all arguments
<distance>	required argument
[, <S>]	optional argument
<cr>	required carriage return

1-3 COMMAND ACCESSIBILITY

Three levels of control are available with the SRS-M1 Small Industrial Robot System. The levels are Interactive {I}, Program {P}, and Manual {M}:

{I}	Interactive	Used for the execution of single commands
{P}	Program	Used for the execution of user programs
{M}	Manual	Used for teaching points through joint moves from teach pendant

Certain commands are only applicable for certain control modes. Chapter 4 describes each command and identifies their applicable modes of control by {I}, {P}, and/or {M}.

1-4 RAPL COMMAND FORMAT

All RAPL command lines must use UPPERCASE characters only. RAPL command lines can contain one or more items: a line number (optional), a command identifier (required), one or more arguments (optional), and a carriage return (required). The format is as follows:

```
[line#] COMMAND [<Argument1>][,...]<cr>
```

Line numbers are used only when entering a command line into a program. Entering a valid numeric character at the start of a command line will cause that line to be inserted into the current program being edited (see Chapter 5). When a number is entered as the first character after the prompt, RAPL is put into the program line entry mode. Whatever is typed after the number will be considered an entry into the program currently being edited. Once this is done, the only way to terminate program line entry is to type a <cr> to enter the line, or an <ABORT> to terminate without entry.

Valid line numbers are from 1 to 65536. Entering a line number larger than 65536 will result in an entry equivalent to: $([line\#] \text{ mod } 65536)$ even though it may be displayed as entered. Entering a line number of 0 will result in an error.

If no line number is entered and the HELP mode is enabled, RAPL completes the command when the operator has entered enough characters to identify it. The number of characters needed depends on whether other commands use the same first letters. For instance, there is currently only one command beginning with the letter Y. Thus, entering a Y at the start of a command line immediately infers the desire to use the YCOMP command. On the other hand, the SHIFT command is not unique (different from the SHIFTA command) until it is completed with a space character.

RAPL will also prompt the user for further information needed to complete the command line (once the command itself is complete) when the HELP mode is enabled. Three typical command line entries are seen below. In these

examples the user input is underlined and bold while the RAPL response is normal:

```
>>SHIFT   "<location> BY" :PICK BY <dX,dY,dZ> :1.0,-.5<cr>
>>APPRO  <location>, PICK. <by a distance of>: 2.5<cr>
>>MOVE   To <destination> : PICK. <S> for straight: S<cr>
```

Notice that RAPL prompts for the correct syntax of the command, including spaces and commas if needed. Arguments, when required, must follow a space, following the RAPL command. Multiple arguments must be separated by commas or spaces. Commas are recommended for improved program readability.

In {I}nteractive or {M}anual mode, RAPL executes the command line as soon as it receives the carriage return <cr>. {P}rogram line entry terminates with the <cr>, but the command is executed only at program run-time.

In {I} or {M}, changing or editing of a command line is permitted only within the current item (the command and each argument are separate items). Once an item is terminated (by a space, comma or <cr>), it can no longer be changed. The only way to edit a command line is to back-space over information entered previously. The ASCII character expected by RAPL is character 7F (hex). In most terminals this character is issued by pressing the or delete key. In the ROBCOMM package, it is issued from the Back Space (<BS>) key.

At any time before receiving the carriage return, a command line can be aborted with an <ABORT> sequence.

When entering a {P}rogram line, any part of the command line can be edited before the <cr> which enters the line into the program memory.

1-5 NAMING CONVENTIONS

RAPL uses the same naming convention for variables (var_name), locations (loc_name) and program names (prg_name).

Names can consist of up to eight alphabetical and numeric characters, however the first character of the name cannot be a number. For example:

```
'A1234567' is legal
'A123'     is legal
'1A'      is illegal
```

For improved readability underline characters (_) may be used. For example:

```
'PICK_1' rather than 'PICK1'
```

RAPL allows for both precision point (these store the robot position in terms of the motor pulses) and cartesian point variables. Precision point names are identified by the special '#' character located in the first

character of the name. Only seven additional characters are then allowed. For example:

'#PICK_1' is a precision point name
'PICK_1' is a cartesian point name

Both location types can co-exist with the same key name (such as PICK_1 and #PICK_1) in the location directory.

Another special form of a location name is the TEACH-type location. With the TEACH mode enabled, pressing the Teach Button on the pendant will store the current robot pose or location. In this case, the name used will consist of a five-character Template and a three-digit Counter. If the template entered is less than five characters, the remaining character spaces will be filled with "under-score" characters:

Template = LOCN, Counter = 1	LOCN_001
Template = PT, Counter = 7	PT__007
Template = #TRAY, Counter = 73	#TRAY073

The last example would be a precision point.

1-6 TOKENIZED FORMAT

The tokenized format of RAPL commands permits program line decoding at a rate of roughly ten times that of the source format. In this format, a number (Token) preceded by a slash is substituted for the english language RAPL command. Due to the speed possible in interpreting the number, RAPL can determine a tokenized command much more quickly.

Tokenized format is recommended for all programs once they are thoroughly debugged. The available ROBCOMM software package for use with IBM PC or compatible computers contains a function to automatically tokenize a program.

Mixing of Tokenized and non-tokenized commands is possible. Programs may be written directly in tokenized form, however this makes program de-bugging more difficult.

Chapter 4 contains a description of the RAPL commands, and the command line syntax. Also shown is the equivalent tokenized command format. APPENDIX A contains a short form list of standard RAPL commands, their functions and tokens.

CHAPTER 2 - PROGRAMMING CONSIDERATIONS

2-1 PROGRAMMING AIDS

Included in the SRS-M1A Small Industrial Robot System resident RAPL is a line oriented editor. Refer to Chapter 5 for additional details.

The optional SRS-RAPL/PC, personal computer software development package (ROBCOMM) is available for advanced programming features such as a RAPL tokenizer and off-line programming.

2-2 HELP MODE

When using the RAPL syntax builder, the programmer need only type enough characters of a command to ensure that it is unique. The system then prompts the user for any further information required. Advanced programmers may find this feature a hinderance an can disable it with the NOHELP or DISABLE HELP command.

2-3 DEFINING THE WORKSPACE

Refer to figure 2-1 for illustration of the SRS-M1 world co-ordinate system. Note that 0,0,0 is at the centre of the base on the ground plane. The arrow indicates the positive (+) direction. The positive directions of the cartesian axes are defined by the "Right-Hand Rule". Using the fingers of the right hand, if the index finger is the positive X axis, the middle finger will line up along the positive Y axis and the thumb along the positive Z axis. The orientation of the wrist is defined by three angles: yaw (orientation about the Z axis), pitch (orientation about the Y axis) and roll (orientation about the X axis). The Right-Hand Rule applies in determining the direction of these angles as well. Point the thumb along the positive axis and the fingers will curl in the direction of positive rotation.

2-3 DEFINING THE WORKSPACE (Continued)

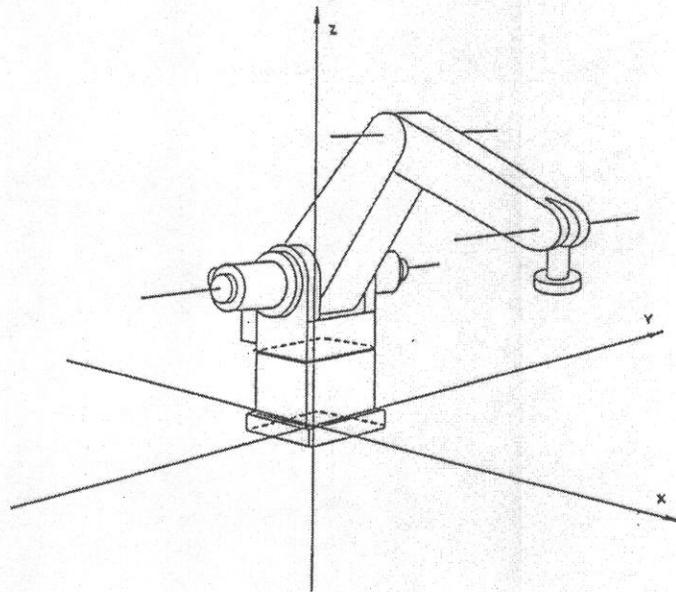


FIGURE 2-1 Defining the workspace - World Co-ordinates

Refer to figure 2-2 for illustration of each of the SRS-M1 five joint motions. The arrow indicates the positive (+) direction.

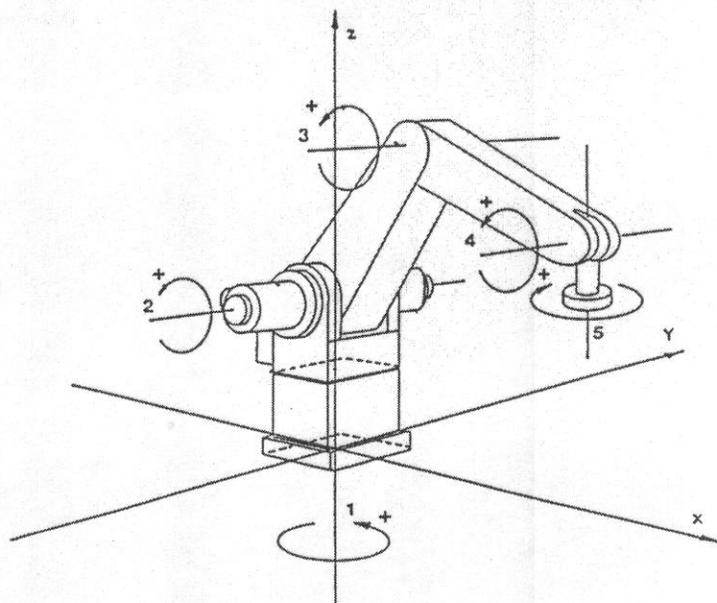


FIGURE 2-2 Defining the workspace - Joint Motions.

2-4 AUTOSTART

The SRS-M1 Small Industrial Robot System provides the user an Autostart feature allowing the robot to be used in production environments without the use of a terminal.

The autostart sequence is activated when the Auto Start switch is depressed as the system power is turned on. In that case, the controller searches its memory for a program called 'AUTO_ST'. If the program is found, it is executed. Refer to appendix 'D' for a sample Auto Start procedure.

2-5 TOOL COORDINATE PROGRAMMING

The TOOL command allows the RAPL user to specify the position of the centre of interest (Tool Centre Point or TCP) on whatever tool the robot is carrying. The TCP is then the position to which all location measurements are made. Straight line moves for instance, will keep the TCP moving linearly at a constant speed, the ALIGN command will keep the TCP fixed in space while re-aligning the wrist pitch to a major axis, and any DEPART or APPRO commands use the tool axis as the direction of the motion relative to the end point.

The tool transform is a cartesian location referenced to the tool coordinate system. This is an orthogonal system with its origin at the centre of the outer surface of the tool flange. The tool coordinate system moves with the tool flange. The X axis is normal to that point. Y and Z axes are aligned with the robot Y and Z axes when the robot is at the READY position (see Figure 2-3). The tool coordinate angles are defined by three angles: yaw (orientation about the Z axis), pitch (orientation about the Y axis) and roll (orientation about the X axis).

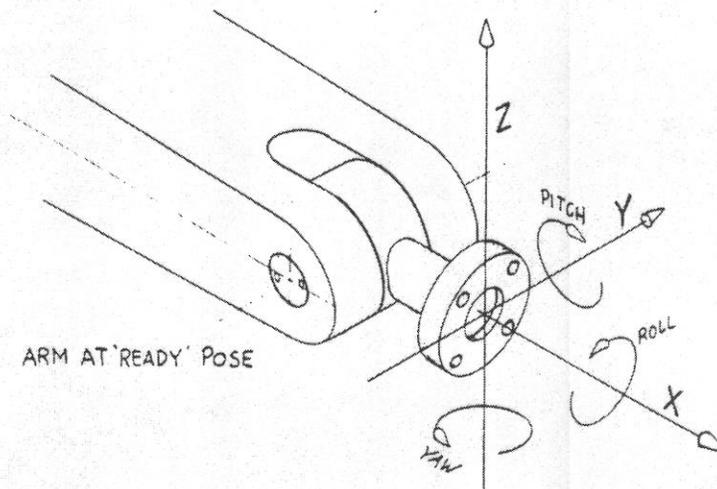


FIGURE 2-3 Orientation of the tool coordinate system.

2-6 MANUAL CONTROL

Manual mode allows the operator to move the robot using the "TEACH PENDANT". RAPL provides two modes of manual control. The first is the JOINT MANUAL mode. In this mode, the 8 toggle switches on the teach pendant, the control each robot joint independently. This is the only possible MANUAL mode until the robot has been homed.

The second mode, which is available after the robot is homed, is CYLINDRICAL MANUAL mode. In this mode, joints 2 and 3 are synchronized so the tool flange will move in a purely radial or vertical (Z axis) direction. This simplifies arm positioning and location teaching.

If fewer than 5 axes are selected, two additional functions are available on the teach pendant during MANUAL mode:

ALIGN: Switch #7 is equivalent to entering "ALIGN" at the keyboard. RAPL will re-position the robot so that the tool centre point is at the same position as before but the tool X axis is aligned with the nearest major axis, vertical or horizontal (see ALIGN in CHAPTER 4).

LIMP: Switch #8 is equivalent to entering LIMP or NOLIMP at the keyboard, depending on which direction the switch is pushed. This opens the servo-loop so a user can move the robot by hand while the controller keeps track of the arm's position. This simplifies rough positioning.

The function of the switches under various situations is seen in TABLE 2-1. For more information, read the MANUAL, ALIGN, LIMP and NOLIMP commands in CHAPTER 4.

SWITCH #	JOINT mode	CYL mode	JOINT mode	CYL mode
	5 - 6 axes	5 - 6 axes	7 - 8 axes	7 - 8 axes
1	JOINT 1	JOINT 1	JOINT 1	JOINT 1
2	JOINT 2	RADIUS	JOINT 2	RADIUS
3	JOINT 3	VERTICAL	JOINT 3	VERTICAL
4	JOINT 4	JOINT 4	JOINT 4	JOINT 4
5	JOINT 5	JOINT 5	JOINT 5	JOINT 5
6	-	-	JOINT 6	JOINT 6
7	ALIGN	ALIGN	JOINT 7	JOINT 7
8	LIMP/NOL.	LIMP/NOL.	JOINT 8	JOINT 8

TABLE 2-1 Switch functions in MANUAL Mode.

3-1 INTRODUCTION

RAPL contains several different types of commands. This Chapter provides a quick overview of those commands separated by type. Prior to using any commands from this list, refer to chapter 4 for the correct syntax and function of each command.

3-2 MOTION COMMANDS

Motion Commands can support up to eight axes of motion control. The robot is programmed to move using the SPEED command at a percentage of the maximum speed of the robot.

There are three types of robot motion programmable through RAPL: Joint interpolated, Straight-line, and Path motion. Most commands will use Joint Interpolated motion (all joints commanded start and stop at the same time). The JOG command is always a straight line move while other commands may use the straight line [S] argument (MOVE, APPRO, or DEPART). In this case, the robot moves using five axis straight line interpolation. In Path motion, the robot creates an eight-axis cubic spline function through a series of programmed locations. The path commands are CTPATH, GOPATH, and CPATH. Constant tool-tip velocity for all path moves is set up using the "ENABLE CARTVEL" command.

An optional software switch provides a faster motion type for joint interpolated moves. This is what is called the SLEW mode and is turned on by the "ENABLE CONSTVEL" command. This mode makes joint interpolated moves much less smooth, but decreases the time required for longer moves. It should not be used for normal motions, but may be used to decrease cycle time if, for instance, long sweeps of joint 1 are required.

Using motion commands LOCK and UNLOCK, the robot controller can control two fully independent machines. One or more axes can be locked together to perform one coordinated motion.

Note that a motion command executed in MANUAL {M} mode will place operator back in Interactive {I} mode.

On power up of the robot controller before the robot has been homed, no absolute motion command, or command which references the cartesian work-space (such as JOG), will be permitted. Any attempt to issue such a command will cause the controller to display the error message "041 NOT HOMED". The relative motion commands JOINT and MOTOR may be used before homing the machine. Take extreme care with these commands (and MANUAL moves) before homing as the joint limit checking does not function until the HOME procedure has been completed.

3-2 MOTION COMMANDS (Continued)

Once the robot has been HOMED, all motion commands are tested for movement outside of software limits prior to the robot starting the motion. These limits are defined by the kinematics of the robot structure. The software limits change depending on the orientation of the robot arm. For example, joint 3 can provide a range of motion which is plus or minus 100 degrees added to the angle of joint 2.

ALIGN	Align the tool with the nearest major axis.
APPRO	Move the robot to an "approach" location.
CPATH	Execute a continuous path.
CIPATH	Program a continuous path comprised of "teach" points.
DEPART	Move the robot away from its current location.
FINISH	Flag instruct a motion command to complete. motion prior to decoding the next program line.
GAIN	Change the positional gain of the servos.
GOPATH	Execute a continuous path programmed with CIPATH.
HALT	Stop all motion or sets up "feed-hold" on input.
HOME	Initialize robot position registers.
JOG	Move robot by a cartesian increment in a straight line.
JOINT	Move a single joint by an angular displacement
LIMP	Disengage positional servos
LOCK	Prevent selected joints from motion
MA	Move all 5 joints to an absolute radian value
MI	Move all 5 joints by an incremental radian value
MOTOR	Drive selected motor by a selected number of pulses
MOVE	Move the robot to a specified location
NOLIMP	Re-engage positional servos
READY	Move arm to 'READY' position.
SPEED	Set speed of robot motion
UNLOCK	Allow selected joints to move

TABLE 3-1 Motion Command Summary

3-3 INPUT/OUTPUT COMMANDS

RAPL input/output commands permit the robot to interface to a number of different external devices. RAPL can access digital input and output ports, dual RS232 communication channels, and optional analog inputs.

The digital inputs can be used to sense switch closure signals in the workplace, providing logical decisions for use in program flow control. The digital outputs can be used to control external equipment.

The Dual serial communication channels provide input and output to a video display terminal, host computers, serial printers or any other serial device. In standard configuration, they conform to the RS-232 standard. Optionally, they can be converted to RS-422.

The optional analog inputs can provide feedback from sensor elements in many robot applications.

ACOUT	Output analog voltage to selected channel.
ANALOG	Read selected analog channel.
ARM	Enable or disables arm power.
CONFIG	Set Configuration of the RS232 ports.
DEVICE	Select the RS232 port for use.
FLASH	Set Flash Interval of light on teach pendant panel.
IFPOWER	Check status of arm power.
IFSIG	Conditional statement based on status of selected input(s).
IGNORE	turn off auto-interrupt feature of robot.
INPUT	input data into user program during run time.
NOFLASH	stop flashing of light on teach pendant panel.
ONPOWER	wait for arm power to come on.
ONSIG	Turn on auto-interrupt feature of robot.
OUTPUT	Turn on selected output(s).
PRINT	Output user information to printer port.
PRINTI	Output variables in an integer format to printer port.
PRINTV	Output variables to printer port.
SERIAL	Display the RS232 port status.
TRIGGER	Change the state of a digital output in path move.
TYPE	Display user information on video display.
TYPEKI	Display variables in integer format on video display.
TYPEV	Display variables on video display.
WAIT	Wait for a selected condition of an I/O port.

TABLE 3-2 Input/Output Command Summary

3-4 GRIPPER COMMANDS

The SRS-M1A robot is equipped for control of a pneumatic, servo (electric) or electromagnetic gripper.

The pneumatic gripper is controlled by the OPEN and CLOSE commands. The air pressure must be manually regulated to provide the desired gripping force.

The servo gripper is controlled in the position mode by the GRIP command. It can be opened or closed in the force mode using the OPEN and CLOSE commands.

The magnetic gripper is controlled by the MAGGRIP command. Gripping force of the electromagnetic gripper is set as a percentage from 1 to 100. The actual gripping force depends on the gain of the magnetic amplifier and the coil type.

CLOSE	Close gripper (at specified force if SERVO).
GRIP	Change position of Servo-Gripper fingers.
MAGGRIP	Select gripping force level for Magnetic gripper.
OPEN	Open gripper (at specified force if SERVO).
TOOL	Specify a tool centre point.
WGRIP	Store the current finger spacing of the SERVO gripper as a variable.

TABLE 3-3 Gripper Command Summary

see Pg. ~~4-144~~ 4-144

3-5 SYSTEM COMMANDS

The RAPL system commands provide general program manipulation, memory allocation, system timer clock, controller status, and the human interface.

Elementary program line editing commands allow the programmer to alter the contents of a program in the memory.

RAPL supports a variety of memory oriented commands intended to provide the user with a working memory space which can be tailored to the specific need.

RAPL also has a HELP feature intended to aid inexperienced users of the system. This assistance takes the form of a "syntax builder". As the command is formulated on the terminal, the operator will only enter enough characters needed to make the command unique before the controller spells out the rest.

?	List Commands.
;	Documentation statement.
ALLOC	Partition and clear robot memory.
COPY	Duplicate program.
DELAY	Provide a time delay.
DELETE	Delete program.
DIR	List program names that are resident in memory.
DISABLE	Turn off software "switch".
EDIT	Enter line editor or create a new program.
ENABLE	Turn on a desired software "switch".
FREE	Display user memory status.
HELP	Turn syntax-building feature on.
LISTP	List program to a selected device.
MANUAL	Enable the teach pendant for movement of robot.
NEW	Clear user memory.
NEXT	Single step through program.
NOHELP	Turn syntax building feature off.
NOMANUAL	Disable the MANUAL mode.
NOTEACH	Disable the teach push-button on teach pendant.
NOTRACE	Disable TRACE mode.
PASSWORD	Permit access to Monitor Level Commands.
RENAME	Change program name.
RUN	Execute a program from memory.
STATUS	Display operating status of robot.
TEACH	Enable the teach pushbutton on teach pendant.
TIME	Read the system timer clock.
TRACE	Line numbers are displayed on terminal as executed.

TABLE 3-4 System Command Summary

3-6 ROBOT LOCATION ASSIGNMENT

One of the most useful capabilities of RAPL is its ability to deal with named robot locations. There are commands which are used to inform the operator of the robot's current location, others to input and modify locations and others to extract dimensional information from location data.

RAPL deals with locations in either inches or millimeters. A switch must be set on the mother board of the RAPL controller to change from one to the other. Changing without re-defining locations is dangerous and can lead to collisions.

There are eight elements used to describe each robot location. The cartesian (XYZ) location of the tool point is defined by its location in space referenced to the centre of the robot base (unless an OFFSET command has been issued). The orientation of the wrist is defined by three angles: yaw (orientation about the Z axis), pitch (orientation about the Y axis) and roll (orientation about the X axis). In addition the six coordinates defined above, RAPL can store the position of the robot on a linear track or on an X-Y gantry in a named location.

The component extraction functions are used to read dimensions from a cartesian location and store the desired component as a variable. The value is returned in real format. The name of the destination variable must be specified for the extracted component.

The ability to define a location in terms of a remote coordinate system may also be useful. This is performed by the OFFSET function. On the other hand, the TOOL function permits changing the tool point at which the robot works if the physical tool changes.

3-6 ROBOT LOCATION ASSIGNMENT (Continued)

ACOMP	The pitch coordinate is stored as a variable name.
ACTUAL	Define a location as the current actual robot position.
COMP	Extract a defined component from a stored location.
DLOCN	Delete a stored location.
HERE	Define a location as the current commanded robot position.
LISTL	List any or all locations stored in memory.
GANTRY	Set up the sixth and seventh axes as an X-Y gantry.
OCOMP	The yaw coordinate is stored as a variable name.
OFFSET	Redefine the base coordinate.
POINT	Define a location.
SET	Equate a new location with an existing location.
SHIFT	Shift a location by an incremental amount in x,y,z.
SHIFTA	Shift a location by an incremental amount in any or all coordinates.
TCOMP	The roll coordinate is stored as a variable name.
TRACK	Set the sixth axis to either an X or Y track.
W0	Display current robot commanded position (Joint and World coordinates).
W1	Continually display actual current robot position (Motor coordinates).
WE1	Continually display actual current position of the robot extra axes (Motor coordinates).
W2	Display the robot actual position.
W3	Continually display the commanded robot position (Motor coordinates).
WE3	Continually display the commanded position of the extra axes (Motor coordinates).
W4	Display the robot next end point.
W5	Continually display the robot position error (Motor coordinates).
WE5	Continually display the position error of the extra axes. (Motor coordinates).
WITH	Permit numbered locations to be accessed using a TEACH mode template. Used for implied location access.
XCOMP	The X coordinate is stored as a variable name.
YCOMP	The Y coordinate is stored as a variable name.
ZCOMP	The Z coordinate is stored as a variable name.

TABLE 3-5 Robot Location Assignment Command Summary

3-7 PROGRAM FLOW FUNCTIONS

RAPL program flow functions permit conditional and un-conditional branches within programs. Line number destinations may be defined by stored variables which permit "case" type programming.

ABORT	Terminate program execution and stop motion.
GOSUB	Control is passed to specified sub-program.
GOTO	Un-conditional branch to a line number.
IF	Branch if variable expression is true.
IFPOWER	Branch on status of arm power.
IFSIG	Branch on status of selected input(s).
IFSTART	Branch on status of autostart switch.
IFSTRING	Branch on status of a string comparison.
ONERR	Trap an error condition.
ONPOWER	wait for arm power to be turned on.
ONSIG	Turn on auto-interrupt feature of robot.
ONSTART	Wait for auto-start switch to be pressed.
PAUSE	Halt program flow until the PROCEED command.
PROCEED	Continues program flow after a PAUSE command.
RETRY	Following an error correction, this command will retry the line.
RETURN	Returns control to calling program.
RUN	Start a program or repeat it a number of times.
STOP	Command to terminate a program.

TABLE 3-6 Programming Flow Command Summary

3-8 STRING FUNCTIONS

The RAPL language can manipulate up to 4 text strings. The strings are identified by the special character &, followed by a number 1 to 4. These strings can be up to 32 characters in length. Strings are bounded by the single quote marks when they are being assigned. A string can consist of any printable character, or even any decimal byte value. For instance:

```
>>! &1 = 'Hello'
```

assigns the string **Hello** into string 1. If the programmer wished to sound a bell after printing this string, he could encode the ASCII bell character, that has a decimal value of 7 into the string by doing this:

```
>>! &1 = 'Hello\7'
```

The back-slash character tells the string interpreter that a decimal code is to be placed in the string. Carriage returns and line feeds can be placed into the strings using this technique. A carriage return is a decimal value 13, while a line feed has a value of 10.

```
>>! &1 = 'Type this, then a new line\13\10'  
>>TYPE &1  
Type this, then a new line  
  
>>
```

Strings can be concatenated, using the standard assignment command !. As an example, type in the following sequence at the terminal:

```
>>! &1 = 'Hello'  
>>! &2 = ' World'  
>>! &3 = &1 + &2  
>>TYPE &3  
'Hello World'  
>>
```

The string number appearing on the left side of the assignment statement cannot appear on the right side in the same statement.

As shown in the above example, the TYPE command can be used to display a string. The operator can also enter a string using the INPUT command as shown here:

```
>>INPUT &1
```

3-8 STRING FUNCTIONS (Continued)

The string entry will be terminated when the operator enters a carriage return. The carriage return will not be part of the stored string value.

Special commands exist that permit encoding of variable-type data into a string, or decoding data from a string. This is useful when interfacing the robot to devices that transmit data in a serial form, or which require special serial commands for operation.

CUT	Remove a portion of the string.
DECODE	Decode a data variable value from a string.
ENCODE	Encode a data variable into a string.
IFSTRING	Compare two strings and branch on result.
PASTE	Paste a new portion of text into a string.
STRPOS	Return the character location of a sub-string.

TABLE 3-7 RAPL String Functions

3-9 MATHEMATICAL FUNCTIONS

RAPL contains a set of commands which can be used to perform mathematical functions on variables stored in memory. The four elementary arithmetic instructions are supported. A library of advanced mathematical instructions are also included.

The advanced mathematical package offers the user a range of extended functions which can be used in the control program. These functions include trigonometric functions and the square root function. The trig functions provide results in either radians or degrees. Obtaining an answer in degrees is useful since they can be directly used in location assignments, such as the SHIFTA command. Radians are useful in determining locations from purely mathematical calculations.

The destination for all mathematical functions is a defined variable name. The arc tangent function is particularly useful, since it provides the angle defined by the two separate elements X and Y. The result is an angle defined by the relationship of $ANGLE = ATAN(Y/X)$.

!	Assign an numeric value to a variable name.
ACOS	Arc cosine.
ASIN	Arc sine.
ATAN	Arc tangent.
COS	Cosine.
DVAR	Delete a variable name.
LISTV	List variables stored in memory.
SIN	Sine.
SQRT	Square root.
TAN	Tangent.

TABLE 3-8 Mathematical Programming Command Summary

3-10 EXTRA AXIS COMMANDS

The SRS-M1A robot controller and the RAPL language provide control of up to eight axes: five are normally used for the M1A arm, and the others may be used to control up to three different servo axes. Two common applications of these extra axes are mounting the robot on a track, and mounting it on a gantry, suspended from above.

To set up the software for use with extra axes, several monitor level commands are used. These are included here for completeness, but are fully documented only in the SRS-M1A/TECH technical manual. Operation of the axes is controlled using "standard" RAPL commands.

In each case, the joint number in the extra axis commands will be numbered 6 through 8. Joints 1 through 5 are robot joints and are not legal inputs to these commands.

MONITOR FUNCTIONS:

@MAXVEL	Enter the maximum possible encoder speed for axis.
@XLIMITS	Enter joint limits for extra axes.
@XPULSERS	Enter number of pulses per turn of extra axis encoder.
@XRATIO	Enter transmission ratio for extra axis.

STANDARD RAPL COMMANDS:

JOINT	Command motion for an axis - extra or robot.
GANTRY	Set up axes 6 and 7 as X and Y gantry axes.
LOCK	Lock an axis against motion commands.
TRACK	Set up axis six as a track - X or Y axis.
UNLOCK	Permit motion of a formerly locked axis.
XCAL	Calibrate an extra axis.
XHOME	Home an extra axis.
XREADY	Move an extra axis to its Zero position.
XZERO	Set the axis current position to zero.

TABLE 3-9 Extra axis commands

3-11 LINE EDITOR

RAPL provides the programmer with a line editing feature which can be used to create or modify existing programs. If a program name is provided, then that program becomes the object of all future edit commands until it is changed. A program cannot be deleted from within the edit mode.

To shorten the development time of a program, RAPL allows the programmer to insert program lines while outside of the standard EDIT mode. While in the {I} or {M} control mode, entering a number before a command identifier will automatically place the following line into the program last edited. No syntax check is performed on the line before it is inserted. A valid line number must be entered, and the target program for the edit mode must have already been defined.

A program line must always start with a line number. A line number is any valid number from 1 to 65535. A line is inserted into a program simply by typing the line number, followed by a command statement. Once a carriage return is entered, the line is complete, and it is entered into the program in the correct numerical sequence. The program memory is updated to reflect the change.

An existing line number cannot be inserted without being deleted first.

Copy	Copy program line to another program line.
Delete	Delete program line.
End	Exit Line Editor.
Insert	Insert program line.
List	Display program line.
Move	Move a program line to another location.

TABLE 3-10 Editor Command Summary

CHAPTER 4 - RAPL COMMANDS

4-1 INTRODUCTION TO COMMAND LIST

This Chapter contains an alphabetical list of RAPL commands describing their syntax, function, modes of operation, any associated hazards, and examples of command usage.

?

FORMATS:

Source: ?
Tokenized: /000

DESCRIPTION:

Displays RAPL commands and their numeric tokens. Monitor level commands will be include in the display only if the correct PASSWORD has been entered.

APPLICABLE MODES:

{I}, {M}

EXAMPLE:

1. Display RAPL commands.

Enter:

>>?<cr>

the response:

000 ?????? 001 ALIGN 002 APPRO 004 DEPART 005 JOG 006 JOINT
etc.

CROSS-REFERENCE:

RAPL COMMANDS: PASSWORD

FORMATS:

Source: [Line#] ; [Message]
Tokenized: [Line#] /098 [Message]

DESCRIPTION:

This command indicates a comment statement. These can be used throughout programs to provide documentation. A space must always follow the ; command.

Although the ; statement is not an active command, it does use memory space, and slows down the operation of the robot program slightly as the comment line must be scanned just like any other. Comments should be used in a concise fashion in order to reduce both effects.

APPLICABLE MODES:

{P}

EXAMPLE:

100 ; This is a comment.

CROSS-REFERENCE:

RAPL COMMANDS: none

OTHER CRS Plus Publications: none

!

FORMATS:

Source: [Line#] ! <&Num>|<Var_name>=<&Num>|'String'|<arg1>[+|-!*|/][arg2]
Token.: [Line#] /070<&Num>|<Var_name>=<&Num>|'String'|<arg1>[+|-!*|/][arg2]

DESCRIPTION:

Permits entry and modification of variable or string data. For variables, the four standard arithmetic operations are supported: '+,-,*,/'. Only one operation can be performed per line. The right hand side arguments can be variable references or explicit constant values. Variable references can be preceded by a plus or minus sign as a unary operator. The assigned value will be treated accordingly.

If the variable on the left hand side of the assignment is previously undefined, a new variable will be created with the value of the operation. If a variable is used on the right side that is previously undefined, a value of 0 is assigned to that variable. Spaces between arguments in the expression are optional.

A special form of this command is useful for incrementing or decrementing variable counters. The command: ! C+ will increment the variable C by one. Similarly, ! C- will decrease the value of C by one.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLES:

1. Set String #1 to 'Transferred WIDGETS':
 >>! &1 = 'Transferred WIDGETS'
2. Set string #2 to String #1:
 >>! &2 = &1
3. A is increased by one to 3:
 >>! A+
4. STACK_S is now equal to -A (-3) multiplied by 4:
 >>! STACK_S = -A * 4

CROSS-REFERENCE:

RAPL COMMANDS: LISTV, DVAR, IF are variable related commands. Also see string commands

ABORT

FORMATS:

Source: [Line#] ABORT
Tokenized: [Line#] /071

DESCRIPTION:

The ABORT statement is an abrupt method of ending a program. If any motion is in progress, it will be stopped immediately. This command is the software equivalent of the ABORT pushbutton on the teach pendant.

APPLICABLE MODES:

{P}

CROSS-REFERENCES:

RAPL COMMANDS: HALT, PAUSE, STOP are other means of stopping the robot and/or a program

ACOMP

FORMATS:

Source: [Line#] ACOMP <LOC_NAME>, <VAR_NAME>
Tokenized: [Line#] /065 <LOC_NAME>, <VAR_NAME>

DESCRIPTION:

The A component (azimuthal or pitch angle) of the specified location is extracted and stored as "VAR_NAME". The angle is stored in degrees.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: XCOMP, YCOMP, ZCOMP, OCOMP, TCOMP, COMP

FORMATS:

Source: [Line#] ACOS <value>, <R!D>, <VAR_NAME>
Tokenized: [Line#] /128 <value>, <R!D>, <VAR_NAME>

DESCRIPTION:

Calculates the angle associated with the value entered. The result is stored in VAR_NAME in either radians <R> or degrees <D>.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: ASIN, ATAN, SIN, COS, TAN

ACTUAL

FORMATS:

Source: [Line#] ACTUAL <LOC_NAME>
Tokenized: [Line#] /152 <LOC_NAME>

DESCRIPTION:

Read the actual position of the robot arm, and store the information as an entry in the location table in memory.

This command differs from the HERE command in that it reads the actual position including any positional error in the arm. Thus, a move to a location taught with ACTUAL will not move to the same location as the one taught: it will have its own error associated with it. ACTUAL is most useful for determining current positional error so that compensation may be made. Refer to the example below.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. A section of a program to determine the vertical position (Z-axis) error at a programmed location (POS1) is as follows:

```
1000 MOVE POS1
1100 FINISH
1200 ACTUAL ERRPOS1
1300 ZCOMP POS1,GOOD_Z
1400 ZCOMP ERRPOS1,BAD_Z
1500 ! Z_ERR = BAD_Z - GOOD_Z
1600 ...
```

CROSS-REFERENCES:

RAPL COMMANDS: HERE

ALIGN

FORMATS:

Source: [Line#] ALIGN
Tokenized: [Line#] /001

DESCRIPTION:

This command provides the user with a method of aligning the tool flange either horizontally or vertically. Essentially, it adjusts joints 4 and 5 so that the tool axis points either horizontally, or vertically downward or upwards. The command is useful in the manual mode, when it is essential to quickly orient the tool correctly.

The manual mode has a hardware version of this command built in. If just 5 axes are installed in the system, switch 7 on the teach pendant performs the equivalent of:

**NOMANUAL
ALIGN
MANUAL**

This feature requires that there not be a command partially entered through the terminal when the switch is depressed.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: MANUAL

ALLOC

FORMATS:

Source: **ALLOC**
Tokenized: /018

DESCRIPTION:

This function clears and re-partitions the robot memory. This can be done with [A] Automatic or [N] Non-automatic partitioning.

If the [A] option is selected, then the memory is automatically divided according to a formula leaving 16 programs in the program table and dividing the remaining memory into 12% variable space, 38% location space and 50% program space. If the [N] option is used, the programmer will be asked to specify the space to be allocated to the program, variable and location tables as well as "Reserved" memory which is used for PATH storage if an expansion memory option has been installed.

WARNINGS:

1. The user memory is always cleared after the ALLOC command is issued.
2. A program table item takes 12 bytes, a location 42 bytes, and a variable 14 bytes of storage. An error will result if not enough memory is available for the user's desired allocation.

APPLICABLE MODES:

{I}, {M}

EXAMPLE:

1. To manually allocate memory (20 programs, 30 variables, 50 locations, leaving 5832 bytes program), Enter:

```
>>ALLOC<cr>  
Are you Sure - (this will erase user memory)? Y  
Total available memory: 08592  
Auto/Nbauto: N<cr>  
# Programs, # Variables, # Locations, # reserved bytes:  
20,30,50,0<cr>
```

CROSS-REFERENCES:

RAPL COMMANDS: FREE, CTPATH, GOPATH, NEW

OTHER CRS Plus Publications: SRS-APP/PATH, SRS-M1A/TECH, SRS-RAM/16PB,
SRS-COMBO.

FORMATS:

Source: [Line#] ANALOG <input#>, <VAR_NAME>
 Tokenized: [Line#] /083 <input#>, <VAR_NAME>

DESCRIPTION:

Up to nine analog input channels are available on the SRS-M1A robot system using the SRS-ANAL/08I option. The SRS-COMBO/32 option increases the number of channels by 16 to 25.

Analog Input channel #9 is the speed selector on the teach pendant. This input can be used for purposes other than MANUAL speed. No expansion option is needed to use this feature.

For all channels of analog in, accuracy of the converters is one part in 256 (8 bit resolution) referred to a voltage of 0 to +5 volts. The input value is an integer ranging between 0 and 255 (255 = +5 Volts). When reading the analog channels, a variable name is used to specify the destination for the data.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. This example inputs an analog value from the teach pendant speed selector and stores the value as the variable ANA_1. This value is then converted into a speed setting as a percentage of full (100%) and used to control the speed of subsequent motion.

```

80 ANALOG 9, ANA_9
81 ! ANA_9 = ANA_9 / 255
82 ! VEL = 100 * ANA_9
84 SPEED VEL

```

CROSS-REFERENCES:

RAPL COMMANDS: AOUT

OTHER CRS Plus Publications: SRS-MAN/COMBO, SRS-MAN/ANAS

ACOUT

FORMATS:

Source: [Line#] ACOUT <CHAN_NUM>, <VAR_NAME>
Tokenized: [Line#] /093 <CHAN_NUM>, <VAR_NAME>

DESCRIPTION:

This command will send a digital value to the selected analog output port. The analog output ports have 8 bit resolution. This function is available only with the expanded I/O option (SRS-COMBO/32).

There are two analog output ports available with this expansion option. These are referred to as 1 and 2 in the <CHAN_NUM> argument.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: ANALOG

OTHER CRS Plus Publications: SRS-MAN/ANAS, SRS-MAN/COMBO

FORMATS:

Source: [Line#] APPRO <LOC_NAME>, <distance>[,S]
 Tokenized: [Line#] /002 <LOC_NAME>, <distance>[,S]

DESCRIPTION:

The APPRO command moves the robot to a position which is the programmed distance back along the tool axis from the stored location "LOC_NAME".

A new location is defined in the direction of the tool axis defined by "LOC_NAME" coordinates and orientation, but offset in the direction of the tool axis by "distance". It then moves the robot to that new location. The location may either be a cartesian point, or a precision point. A positive "distance" implies that the tool will stop in front of the specified location.

The APPRO command enables the programmer to define an approach point without using extra memory resources.

The straight line [S] specifier defines whether or not the approach should be made in a straight line or not. Typically, in an insertion operation, a non-straight approach is made to the approach point, then a straight move command is used to move to the location required.

APPLICABLE MODES:

{I}, {P}

EXAMPLES:

1. Approach point PICK_1 stopping +2 inches away. Decrease speed to 20, then move to point PICK_1 in a straight line and close the gripper.

```
100 SPEED 100
110 APPRO PICK_1,2
120 SPEED 20
130 MOVE PICK_1,S
140 FINISH
150 CLOSE
```

2. Approach point PICK_1 in a straight line stopping +2 inches away.

```
110 APPRO PICK_1,2,S
```

CROSS-REFERENCES:

RAPL COMMANDS: DEPART, TOOL, MOVE

ARM

FORMATS:

Source: [Line#] ARM <ON;OFF>
Tokenized: [Line#] /142 <ON;OFF>

DESCRIPTION:

Sets the status of the Arm Power Relay (APR). If the "ON" parameter is chosen, the APR is enabled and pressing the ARM POWER switch on the front panel will then turn the power on.

If the arm power is on, the ARM OFF command will disable the APR and the arm power will be turned off. This will result in an ARM POWER error. The switch on the panel will be disabled until the ARM ON or ENABLE ARM command is issued.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: ENABLE, DISABLE, LIMP, NOLIMP

OTHER CRS Plus Publications: SRS-M1A/TECH

ASIN

FORMATS:

Source: [Line#] ASIN <value>, <R|D>, <VAR_NAME>
Tokenized: [Line#] /127 <value>, <R|D>, <VAR_NAME>

DESCRIPTION:

Calculates the angle associated with the value entered. The result is stored in VAR_NAME in either radians <R> or degrees <D>.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: ACOS, ATAN, SIN, COS, TAN

ATAN

FORMATS:

Source: [Line#] ATAN <X>, <Y>, <R;D>, <VAR_NAME>
Tokenized: [Line#] /129 <X>, <Y>, <R;D>, <VAR_NAME>

DESCRIPTION:

Determines the angle associated with the expression of angle = ATAN(Y/X).
The angle will be determined in the range of 0 to 360 degrees, or 0 to 2 pi radians.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: ACOS, ASIN, SIN, COS, TAN

CLOSE

FORMATS:

Source: [Line#] **CLOSE** [,torque]
Tokenized: [Line#] /038 [,torque]

DESCRIPTION:

With a pneumatic gripper installed, this command will close the gripper by switching the solenoid in the arm. With the SERVO GRIPPER installed, it will close the gripper at a specified torque. If no torque argument is specified, then the previous torque value (entered with either the OPEN or CLOSE command) will be used.

When in Manual mode the gripper can be closed by use of the gripper switch on teach pendant.

WARNING

1. The OPEN, CLOSE and GRIP commands have been re-structured with RAPL version 4.10 and greater. Older versions of RAPL used the OPEN and CLOSE commands only with the air gripper option. The OPEN and CLOSE commands now control the servo gripper in the force mode of operation. This change may lead to compatibility problems with programs written for RAPL versions prior to 4.10.

APPLICABLE MODES:

{I}, {P}

EXAMPLE:

1. To close an air gripper:
>>CLOSE<cr>
2. To close the SERVO gripper at a torque of 60% of full:
>>CLOSE 60<cr>

CROSS-REFERENCES:

RAPL COMMANDS: OPEN, GRIP, @@GTYPE

OTHER CRS Plus Publications: SRS-M1A/TECH, SRS-MAN/SGRIP

COMP

FORMATS:

Source: [Line#] COMP <COMP_NUM>, <LOC_NAME>, <VAR_NAME>
Tokenized: [Line#] /181 <COMP_NUM>, <LOC_NAME>, <VAR_NAME>

DESCRIPTION:

This command replaces the series of commands XCOMP, YCOMP, ZCOMP, OCOMP, ACOMP, TCOMP and adds the ability to extract components from precision points as well. There are also two extra components associated with any cartesian location in the case when a TRACK or GANTRY is specified. In this case, the position of these extra axes can also be extracted and stored in a variable.

A notable difference in the COMP command is that angular cartesian component values will be extracted in radians, while the ACOMP, OCOMP and TCOMP extract degree values.

APPLICABLE MODES:

{I}, {P}, {M}

EXAMPLE:

1. Extract the Z component of a location:
>>COMP 3,POINT1,P1_Z<cr>
2. Extract the Joint 5 component of a precision point:
>>COMP 5,#KIT_024,TK24<cr>
3. Extract the X Gantry component of a cartesian location (valid only after execution of the GANTRY command):
>>COMP 7,PICK_G1,X_G_POS

CROSS-REFERENCES:

RAPL COMMANDS: XCOMP, YCOMP, ZCOMP, OCOMP, ACOMP, TCOMP

FORMATS:

Source:

[line#] CONFIG <dev#>, <baud>, <parity>, <#data>, <#stop>, <handshake>, <echo>

Tokenized:

[line#] /110 <dev#>, <baud>, <parity>, <#data>, <#stop>, <handshake>, <echo>

DESCRIPTION:

This command alters the configuration of the serial ports (DEVICE #0 and DEVICE #1) to match other equipment requirements. The following are the only allowable values for each argument:

Device Number:	0,1
Baud rates:	50,75,110,135,150,200,300,600,1200,1800,2400,4800, 7200, 9600,19200
Parity Check:	[E] Even, [O] Odd, [N] None
Data Bits:	5,6,7,8
Stop Bits:	[1] 1 bit, [2] 1 1/2 bits, [3] 2 bits
Handshake:	[R] RTS/CTS, [X] XON/XOFF, [N] None, [B] Both
Echo:	[E] ON, [N] OFF

Devices #0 and #1 are initialized on start-up to the configuration last specified by the user.

WARNINGS:

When issuing this command, all arguments must be entered and all must be separated by commas.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLES:

1. This example will set the configuration of device #1 to 1200 baud, even parity, 8 data bits, 1 stop bit, XON/XOFF handshake, and no echo:

```
>>CONFIG 1,1200,E,8,1,X,N<cr>
```

CROSS-REFERENCES:

RAPL COMMANDS: SERIAL

OTHER CRS Plus Publications: SRS-M1A/TECH

COPY

FORMATS:

Source: COPY <OLD_NAME>, <NEW_NAME>
Tokenized: /099 <OLD_NAME>, <NEW_NAME>

DESCRIPTION:

Often it is necessary to utilize a modified version of an existing program. In order to use the old program for this purpose, and yet retain this original version, it is necessary to duplicate the program using this COPY command.

WARNINGS:

Errors will result if there is insufficient memory space to contain the new program.

APPLICABLE MODES:

{I}, {M}

EXAMPLE:

1. Copy program TEST to a new file called TEST1.
>>COPY TEST,TEST1<cr>

CROSS-REFERENCES:

RAPL COMMANDS: RENAME, DELETE, EDIT

FORMATS:

Source: [Line#] COS <angle>, <R|D>, <VAR_NAME>
Tokenized: [Line#] /125 <angle>, <R|D>, <VAR_NAME>

DESCRIPTION:

Will determine the Cosine of the given angle and store result in radians <R> or degrees <D> as a VAR_NAME.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: SIN, TAN, ACOS, ASIN, ATAN

CPATH

FORMATS:

Source: [Line#] CPATH <LOC_NAME>[,LOC_NAME,...]
Tokenized: [Line#] /163 <LOC_NAME>[,LOC_NAME,...]

DESCRIPTION:

This command permits the robot to move through a random series of points. In this command, unlike the CTPATH command, the locations specified in the argument list can be any arbitrary location. The location must be defined, and all the locations in the list must be either cartesian locations, or precision points. No mixing is allowed. Up to 16 locations are permitted in the argument list.

Triggers will be activated during the path execution if the TRIGGER table includes activities at any of the locations specified in the CPATH argument string and the TRIGGER flag has been ENABLED.

WARNINGS:

No two consecutive locations in the list may be identical. This will produce a mathematical error and processing will be aborted.

If more than 5 axes are used in the controller, precision points must be used.

APPLICABLE MODES:

{I}, {P}

EXAMPLE:

1. To move through a series of points without stopping. Any single location can be used, so long as it does not appear consecutively.

```
>>CPATH A,B,C,B,C,A,B
```

2. This will produce an error, since the location B appears consecutively. Also, the precision point #EE is not allowed since the remainder of the locations in the list are cartesian locations.

```
>>CPATH A,B,B,C,S,#EE
```

CROSS-REFERENCES:

RAPL COMMANDS: ENABLE, DISABLE, CTPATH, GOPATH

OTHER CRS Plus Publications: SRS-M1A/TECH, SRS-APP/PATH

FORMATS:

Source:

[Line#] CTPATH <template>, <start index>, <end index>, <speed>

Tokenized:

[Line#] /164 <template>, <start index>, <end index>, <speed>

DESCRIPTION:

This command will generate a continuous path through the series of points that were previously entered via the TEACH mode as described in the argument list. The location list is presented in terms of the teach indices that bound the intermediate locations. The command line contains the teach template used to program the points specified. The speed argument is an additional argument that permits the programmer to design a path that is independent of the current robot speed setting. Thereafter, when the GOPATH command is executed, the robot moves in the joint interpolated mode to the first location in the path at current speed and then executes the path at the speed specified in the CTPATH command.

CTPATH has several important differences from the CPATH command. First, the CTPATH command does not actually execute a path. It simply determines the path parameters for later execution. A subsequent GOPATH command will execute the path.

Second, the CTPATH command obtains locations from an array of locations taught in the TEACH mode. The CTPATH command can therefore program a path through up to 256 locations if there is sufficient memory to store the path data. The primary advantage is that subsequent GOPATH commands will not have a time delay associated with calculating the path parameters.

Third, the robot will make a joint interpolated move to the starting "knot" of the CTPATH and then will execute the path. The path will not include the position where the robot is when the GOPATH command is executed.

The CTPATH command will utilize the trigger table contents in the same fashion as the CPATH command.

WARNINGS:

1. In the location list, no two consecutive locations may be identical. This condition would produce a math error. For this reason, it is wise to move the robot through the location list in order to verify the path locations.
2. Due to the high memory requirements of the CTPATH command, it requires the installation of the expanded memory option.

Continued...

CTPATH (Cont)

APPLICABLE MODES:

{I}, {P}

EXAMPLE:

1. Having issued the command:

```
>>TEACH A,0
```

Assuming that locations A___000 through to A___010 were stored using the teach mode, we can now calculate a path through those points, at a speed of 50% of full.

```
>>CTPATH A,0,10,50
```

The reverse path can also be executed by reversing the indices

```
>>CTPATH A,10,0,50
```

Variables may be used in the CTPATH command argument list as below:

```
>>CTPATH &1,FIRST,LAST,SPEED
```

The above command will use string #1 as the template, and the variables **FIRST**, **LAST**, and **SPEED** as arguments in the list.

CROSS-REFERENCES:

RAPL COMMANDS: ENABLE, DISABLE, GOPATH, CPATH, TRIGGER

OTHER CRS Plus Publications: SRS-M1A/TECH, SRS-APP/PATH

CUT

FORMATS:

Source: [Line#] CUT <STR_NUM>, <CHAR_INDEX>, <NUM_CHARS>
Tokenized: [Line#] /145 <STR_NUM>, <CHAR_INDEX>, <NUM_CHARS>

DESCRIPTION:

This command will take the specified string, and will remove **NUM_CHARS** of characters from the string, starting with, and including the character indicated by **CHAR_INDEX**.

APPLICABLE MODES:

{I}, {P}, {M}

EXAMPLE:

1. Assign a value to string 1, and remove 3 characters from it starting at index 5.

```
>>! &1 = 'LET US CHOP THIS LINE'  
>>CUT 1,5,3  
>>TYPE &1  
LET CHOP THIS LINE  
>>
```

CROSS-REFERENCES:

RAPL COMMANDS: PASTE, STRPOS, !, ENCODE, DECODE, IFSTRING are other string related commands.

DECODE

FORMATS:

Source: [Line#] DECODE <STR_NUM>, <CHAR_INDEX>, <VAR_NAME>
Tokenized: [Line#] /147 <STR_NUM>, <CHAR_INDEX>, <VAR_NAME>

DESCRIPTION:

Will decode an ASCII encoded real number from the specified string number, starting at the specified character index (character position within the string). The character index takes on a value from 1 to 31.

WARNING

The letter 'E' may be a legal character as part of a number encoded in a string, as in '1E-6' (one times ten to the minus six). Thus an 'E' following a number without a space will be decoded as if it has an exponent. Use caution with this letter.

APPLICABLE MODES:

{I}, {P}, {M}

EXAMPLE:

1. Assigning string 1 to the value below, we can extract the real number 510 and place it into the variable AA

```
>>! &1 = 'VALUE IS 510'  
>>DECODE 1,10,AA  
>>TYPEV AA  
510.0000  
>>
```

CROSS-REFERENCES:

RAPL COMMANDS: CUT, PASTE, STRPOS, !, ENCODE, IFSTRING are other string related commands.

DELAY

FORMATS:

Source: [Line#] DELAY <time>
Tokenized: [Line#] /081 <time>

DESCRIPTION:

This function provides a time delay for the robot. The delay is programmable in seconds, and accuracy in excess of 30 milliseconds can be expected. A maximum delay of 65.530 seconds can be programmed per command. Longer delays must use successive DELAY commands.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: TIME, FINISH (may be required after move, before DELAY)

DELETE

FORMATS:

Source: **DELETE <PRG_NAME>**
Tokenized: **/100 <PRG_NAME>**

DESCRIPTION:

Delete program from directory.

APPLICABLE MODES:

{I}, {M}

CROSS-REFERENCES:

RAPL COMMANDS: **COPY, EDIT, RENAME**

FORMATS:

Source: [Line#] DEPART <distance>[,S]
 Tokenized: [Line#] /004 <distance>[,S]

DESCRIPTION:

The DEPART command moves the robot from its present location by plus/minus "distance" in the tool axis direction. The sign of the distance is the same as that for the APPRO command.

The DEPART command with the optional straight line [S] specifier is useful when removing the tool along a straight line in order to eliminate the possibility of collision in tight spaces.

APPLICABLE MODES:

{I}, {P}

EXAMPLES:

1. Approach point PICK_1 stopping +2 inches away. Decrease speed to 20, then move to point PICK_1, Close gripper, delay for 2 seconds, then depart from PICK_1 stopping +2 inches away.

```
100 SPEED 100
110 APPRO PICK_1,2
120 SPEED 20
130 MOVE PICK_1
140 CLOSE
150 DELAY 2
160 DEPART 2
```

2. Depart from PICK_1 in a straight line stopping +2 inches away.

```
160 DEPART 2,S
```

CROSS-REFERENCES:

RAPL COMMANDS: APPRO, TOOL (Affects the depart direction)

DEVICE

FORMATS:

Source: [Line#] **DEVICE** <dev#>
Tokenized: [Line#] /112 <dev#>

DESCRIPTION:

The user can select the device for standard communication operations. This facility is normally used for communicating to an auxiliary serial device in the work cell other than the standard terminal device.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. This example prints a status report to device #1. This could also be done using PRINT-type commands and no DEVICE statement.

```
340 DEVICE 1
350 TYPE 'Production Report'/
355 TYPE 'Widget transferred : '
356 TYPEV W_CNT
357 TYPE ' ' /
```

CROSS-REFERENCES:

RAPL COMMANDS: CONFIG, SERIAL DIR, LISTL, LISTP, LISTV

DIR

FORMATS:

Source: [Line#] DIR [1]
Tokenized: [Line#] /102 [1]

DESCRIPTION:

The directory command will list the names of all programs currently in memory. The program name and its length in characters are displayed on the output device.

Output can be routed to the printer port by using the [1] optional argument.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: EDIT, DELETE, LISTP

DISABLE

FORMATS:

Source: [Line#] **DISABLE** [ITEM]
Tokenized: [Line#] /177 [ITEM]

DESCRIPTION:

This command turns off the corresponding system parameter. The **DISABLE** command operates on a list of system parameters defined in the following list:

TRIGGER	Prevent trigger outputs from being performed.
OUTPUT	Prevent all digital outputs from changing according to the OUTPUT command.
ONSIG	Turn off the ONSIG trigger, but maintain the ONSIG condition.
EDIT	Disable any further modifications to stored programs.
ARM	Disable the ARM power signal, turning off the motor amplifiers.
LOFB	Disable the loss of feedback/collision detection check. This is not recommended.
TRACE	Disable the TRACE mode. Same as NOTRACE command.
FLASH	Turn off the FLASH command. Same as NOTRACE .
APC	Disable the Arm power check. Same as @NAPC monitor command.
HELP	Turn off the HELP mode. Same as NOHELP .
TEACH	Turn off the teach facility, but maintain the TEACH template, and the current value of the teach counter. Same as the NOTEACH command.
MANUAL	Disable the manual mode, but maintain the manual mode type for the next invocation of the MANUAL mode.
CARTVEL	Use joint (not CARTesian) distances to calculate path knot timing.
SLEW	Turn off SLEW (trapezoidal profile) in Joint-Interpolated motion (see Section 3-2).
HOLD	Disable the HOLD feature without losing " HALT ON... " input number.

If no argument is entered, then the parameter list is displayed on the terminal, along with the current parameter setting.

Several of the items in the parameter list are covered by other existing **RAPL** commands. The **HELP/NOHELP**, **TEACH/NOTEACH** commands are examples. The **DISABLE** command will be the standard **RAPL** parameter control function for all future versions. Thus instead of typing **NOFLASH**, the user should type **DISABLE FLASH**. It is recommended that the **RAPL** programmer follows this convention.

Continued...

DISABLE (Cont)

APPLICABLE MODES:

{I}, {P}, {M}

CROSS-REFERENCES:

RAPL COMMANDS: ENABLE, NOHELP, NOMANUAL, NOTRACE, NOFLASH, EDIT,
 NOTEACH, CTPATH, @NAPC, @LOFB

OTHER CRS Plus Publications: SRS-M1A/TECH, SRS-APP/PATH

DLOCN

FORMATS:

Source: [Line#] DLOCN <LOC_NAME>[,.....]
Tokenized: [Line#] /048 <LOC_NAME>[,.....]

DESCRIPTION:

Stored locations are erased permanently from memory.

WARNINGS:

If deleting more than one location, be aware that a location is deleted as soon as the comma following its name is entered. Using the Back-Space function to edit the command is not possible after the comma has been typed.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Deletes stored location 'A'.
>>DLOCN A
2. Deletes stored precision point location '#A'.
>>DLOCN #A
3. Deletes stored locations 'A', 'B', 'C'.
>>DLOCN A,B,C

CROSS-REFERENCES:

RAPL COMMANDS: LISTV, HERE, POINT

FORMATS:

Source: [Line#] DVAR <VAR_NAME>[,...]
Tokenized: [Line#] /067 <VAR_NAME>[,...]

DESCRIPTION:

This command will delete a variable or a list of references from the variable table. When a variable is cleared, the next time it is referenced it returns with a value of 0.

WARNINGS:

If deleting more than one variable, be aware that a variable is deleted as soon as the comma following its name is entered. Using the Back-Space function to edit the command is not possible after the comma has been typed.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Deletes variable HEIGHT.
>>DVAR HEIGHT
2. Deletes variables A,B,C,D,E and F.
>>DVAR A,B,C,D,E,F

CROSS-REFERENCES:

RAPL COMMANDS: !, LISTV

EDIT

FORMATS:

Source: EDIT [PRG_NAME]
Tokenized: /101 [PRG_NAME]

DESCRIPTION:

RAPL provides the programmer with a line editing feature which can be used to create or modify existing programs. The EDIT command activates the line editor mode. When a program name is entered with the edit command, that program then becomes the target program for any directly entered program lines even after leaving the editor. If no program by that name exists, RAPL will create one. An error will occur if the table is full. If no program name specifier is used, the last edited program will be used. A program cannot be deleted from within the edit mode.

To shorten the development time of a program, RAPL allows the programmer to insert program lines while outside of the standard EDIT mode. While in the monitor mode, entering a number before a command identifier will automatically place the following line into the program last edited. No syntax check is performed on the line before it is inserted. A valid line number must be entered, and the target program for the edit mode must have already been defined. An existing line may not be deleted or altered by entering a new line with the same number. To alter an existing line, the old line must first be deleted in the line edit mode.

All editing features can be prevented by the DISABLE EDIT command. EDIT will stay disabled until either the ENABLE EDIT command is issued or a TEACH START is done.

APPLICABLE MODES:

{I}, {M}

CROSS-REFERENCES:

RAPL COMMANDS: LISTP, DELETE, ENABLE, DISABLE. Also refer to Chapter 5 of this manual.

FORMATS:

Source: [Line#] ELBOW <UP;DOWN>
Tokenized: [Line#] /169 <UP;DOWN>

DESCRIPTION:

Specify the position of the elbow (joint #3) for all successive cartesian transformations. Transformations are calculated whenever the programmer requests a move to a cartesian location. This transformation process determines the joint angles required to achieve that location.

The ELBOW command chooses an acceptable configuration for joint #3. The normal elbow configuration is UP when the robot is mounted on a platform. It is normally elbow DOWN if the robot is suspended from a gantry in an inverted fashion. Elbow DOWN is also used whenever the robot is attempting to reach the work space "over its head".

The ELBOW command is typically used in conjunction with the REACH command.

APPLICABLE MODES:

{I}, {P}, {M}

EXAMPLE:

10 ELBOW UP
20 MOVE A

CROSS-REFERENCES:

RAPL COMMANDS: INVERT, REACH

ENABLE

FORMATS:

Source: [Line#] **ENABLE** [ITEM]
Tokenized: [Line#] /176 [ITEM]

DESCRIPTION:

The enable command turns on the corresponding system parameter. The **ENABLE** command operates on a list of system parameters defined in the following list:

TRIGGER	Permit trigger outputs to be performed.
OUTPUT	Permit all digital outputs according to the OUTPUT command.
ONSIG	Turn on the ONSIG trigger, using the existing ONSIG condition.
EDIT	Enable all future EDIT operations.
ARM	Enable the ARM power signal, enabling the motor amplifiers.
LOFB	Enable the loss of feedback/collision detection check. This is highly recommended.
TRACE	Enable the TRACE mode. Same as TRACE command.
FLASH	Turn on the FLASH command using the previous value of the FLASH interval.
APC	Enable the Arm power check. Same as @APC monitor command.
HELP	Turn on the HELP mode. Same as HELP.
TEACH	Turn on the teach facility, using the previous TEACH template, and the current value of the teach counter.
MANUAL	Enable the manual mode, under whatever mode type that was previously.
CARTVEL	Use CARTesian distances to calculate path knot timing.
SLEW	Turn on SLEW (trapezoidal profile) for Joint-Interpolated motion (see Section 3-2).
HOLD	Enable the HOLD feature for previously defined "HALT ON..." input number.

If no argument is entered, then the parameter list is displayed on the terminal, along with the current parameter setting.

Several of the items in the parameter list are controlled by other existing RAPL commands. The **HELP/NOHELP** and **TRACE/NOTRACE** commands are examples. The **ENABLE** command will be the standard RAPL parameter control function for all future versions. Thus instead of typing **TRACE** the user should type **ENABLE TRACE**. It is recommended that the RAPL programmer follows this convention.

Continued...

ENABLE (Cont)

APPLICABLE MODES:

{I}, {P}, {M}

CROSS-REFERENCES:

RAFL COMMANDS: DISABLE, HELP, MANUAL, TRACE, FLASH, EDIT, TEACH,
 CTPATH, @APC, @LOFB

OTHER CRS Plus Publications: SRS-M1A/TECH, SRS-APP/PATH

FORMATS:

Source: [Line#] FINISH
 Tokenized: [Line#] /010

DESCRIPTION:

The FINISH flag instructs the currently executing motion to finish before the next program line is decoded and executed. The FINISH command only acts on the current motion underway.

When FINISH is not specified, the next command will be executed as soon as the path parameters of the commanded motion path have been determined (generally shortly after a motion command has started). Usually, this is ideal since the program will run much faster. Sometimes, however, it is important to synchronize program control with robot motion. As an example, the robot must usually reach its final destination before the signal to close the gripper is issued.

If a program includes two consecutive motion commands, the first command will be completed before the next is processed. No finish statements are needed between consecutive motion blocks. The only exception is the MOTOR command: consecutive MOTOR commands for different axes will not wait for each other to finish.

APPLICABLE MODES:

{P}

EXAMPLE:

1. Without FINISH: This example will result in a collision between the gripper and the object at PICK_1. The CLOSE command in line 150 will be processed shortly after the MOVE command in line 140 has started.

```
110 APPRO PICK_1,2
130 MOVE PICK_1
140 CLOSE
```

2. With FINISH: By inserting a FINISH command between the MOVE and the CLOSE, the arm will reach PICK before the CLOSE signal is sent.

```
110 APPRO PICK_1,2
130 MOVE PICK_1
135 FINISH
140 CLOSE
```

CROSS-REFERENCES:

RAPL COMMANDS: OPEN, CLOSE, DELAY, OUTPUT, IFSIG may require FINISH.

FLASH

FORMATS:

Source: [Line#] FLASH <interval>
Tokenized: [Line#] /084 <interval>

DESCRIPTION:

The READY light on the teach pendant can be used as a general purpose output for use by the programmer.

The flash interval number has a valid range of 1 to 255. Each unit of the flash interval is approximately equal to 20 milliseconds.

If no flash interval value is entered, the default is the previous setting.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Flashes the teach pendant light at an interval of 60 milliseconds.

>>FLASH 3

CROSS-REFERENCES:

RAPL COMMANDS: ENABLE, NOFLASH, DISABLE

FORMATS:

Source: [Line#] ~~FREE~~ [1]
Tokenized: [Line#] /023 [1]

DESCRIPTION:

The FREE command will display the status of the user memory, along with the remaining memory to be used. The optional argument will route the information to the printer port.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Display Memory remaining, Enter:

>>FREE

the response will be:

ROBOT MEMORY ALLOCATION:
NUMBER OF EMPTY VARIABLES : 085
NUMBER OF EMPTY LOCATIONS : 077
NUMBER OF EMPTY PROGRAMS : 009
PROGRAM MEMORY REMAINING : 03659
RESERVED MEMORY ALLOCATED : 00000

CROSS-REFERENCES:

RAPL COMMANDS: ALLOC

GAIN

FORMATS:

Source: [Line#] GAIN <MOTOR #>, <VALUE>
Tokenized: [Line#] /150 <MOTOR #>, <VALUE>

DESCRIPTION:

The GAIN command allows the programmer to soften the response of the robot arm in a predictable and controllable manner.

Although the servo control loop of the robot is complex, and mostly beyond the reach of the operator, the gain command allows the operator to control the proportional gain of the servo loop. Basically, the proportional gain creates a driving voltage to the motor amplifiers proportional to the amount of positional error that the controller sees when comparing the commanded position with the actual position as provided by the encoder feedback.

Since increasing the proportional gain can lead to instability of the servo mechanism, and possible mechanical damage, the operator should keep the GAIN value for any of the robot joints to less than or equal to one (1). The range of values acceptable in the command vary between zero (0) and two (2).

Nominally, the robot position gain factor is 1 on every axis, and this value should be maintained for optimum robot performance.

The programmer can select any motor gain by specifying a number between 1 and 8, or all motors can be selected for a gain change by entering a number 0 for the motor number.

WARNINGS

A low gain will cause the arm to become less stiff. Under load it will droop and paths will not be followed as closely.

Increasing the gain by more than 0.1 at a time may cause a noticeable jump in the arm position. This effect will be more noticeable under load conditions.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

OTHER CRS Plus Publications: SRS-M1A/TECH, SRS-APP/EXTRA

FORMATS:

Source: [Line#] GANTRY
Tokenized: [Line#] /180

DESCRIPTION:

This command sets up a GANTRY condition. This means that every cartesian location includes the position of axes 6 and 7 as X and Y gantry coordinates.

The coordinates of the gantry are stored as a real numbers in 8 bytes immediately above the standard six coordinates of each cartesian entry in the location table. The real values are the "joint" engineering unit values of the axes when the HERE command is issued. Use of @XRATIO, @XLIMITS, and @XPULSES for both axes 6 and 7 is mandatory before issuing this command. Before storing values or commanding motion after the GANTRY command is issued, an XHOME command must have been issued for both axes.

The coordinates stored for the extra axes are entered only as an input to the motion command. Shifting of these coordinates is possible using the SHIFTA command. Entry of a point "off-line" using the POINT command will permit entry of the axis values. All motion commands referring to cartesian locations after issuing the GANTRY command will coordinate with the GANTRY axes.

For long moves involving the gantry axes it may be an advantage to use the SLEW mode. This is entered with the ENABLE CONSTVEL command.

GANTRY is reset using the "TRACK RESET" command.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: TRACK, @XPULSES, @XRATIO, @XLIMITS, SHIFTA

OTHER CRS Plus Publications: SRS-APP/EXTRA

GOPATH

FORMATS:

Source: [Line#] GOPATH
Tokenized: [Line#] /165

DESCRIPTION:

This command will execute a CTPATH that has been entered. Once a CTPATH has been executed, then the path remains in memory until the controller is shut off, or another CTPATH is entered.

When GOPATH is issued, the robot will perform a joint interpolated move to the starting knot of the path at the current robot speed setting and will then start the continuous path at the programmed speed.

APPLICABLE MODES:

{P}, {I}

EXAMPLE:

1. To execute a previously defined path in the Immediate or from the manual mode:

```
>>GOPATH
```

2. From a program:

```
100 CTPATH PNTS,1,35,70
```

```
....
```

```
500 GOPATH
```

The actual path followed was entered with the last CTPATH command. A valid path must be programmed first.

CROSS-REFERENCES:

RAPL COMMANDS: CTPATH, ENABLE, DISABLE

OTHER CRS Plus Publications: SRS-APP/PATH

FORMATS:

Source: [Line#] GOSUB <PRG_NAME> [parameter1][,...]
Tokenized: [Line#] /074 <PRG_NAME> [parameter1][,...]

DESCRIPTION:

Control is passed to the specified sub-program. The sub-program must have a RETURN statement in it so the processor's stack does encounter any problems. The RAPL processor contains a stack which stores the information needed to return to the calling program. The stack allows for up to 10 nested sub-program calls.

The GOSUB command can transfer parameters to the subroutine. These parameters are 8-byte string registers which can contain names of locations, variables etc. for use inside the subroutine. In the subroutine, they are recalled by substituting a special variable name identified as '%0', '%1', .. '%7' depending on their position in the list in the calling line.

There can be 8 parameters in all. The programmer may nest subroutines which make use of the same parameters.

The sub-program name can be entered as a reference to a string. This permits a 'BATCH' approach to program execution, where the operator can enter a program name as a response to a query from the system. See example #2.

APPLICABLE MODES:

{P}

Continued....

GOSUB (Cont)

EXAMPLE:

1. Program MAIN is the executive. Sub-program APPROACH will approach a location by some distance with the ONSIG condition activated. Note that Sub-program E_STOP uses the same parameters as APPROACH did without them needing to be re-defined.

```
50 ; PROGRAM MAIN
60 ;
100 GOSUB APPROACH POINT1,4
110 ...
```

```
PROGRAM APPROACH:
100 ONSIG 4 E_STOP
110 APPRO %0,%1
120 FINISH
130 IGNORE
140 RETURN
```

```
PROGRAM E_STOP:
100 HALT
110 WAIT -4
120 APPRO %0,%1
130 RETURN
```

2. Batch control over program execution can be an effective way to alter robot programming for new jobs. Several tasks can be loaded into the robot memory concurrently, providing decreased downtime required to load new data. Program execution can then be branched to the correct job by using a string to refer to the sub-program.

```
10 TYPE 'ENTER THE JOB NAME (SWITCH, PLATE, KNOB) '
10 INPUT &1
16 TYPE &1
17 TYPE ' NOW BEING EXECUTED' /
20 GOSUB &1
99 GOTO 10
```

The above MAIN program will permit branching to one of three sub-programs. They must be called SWITCH, PLATE or KNOB. If the operator enters a name which does not match any of the selections, a 'PROGRAM NOT FOUND' error will be created at line 20. If necessary, testing for correct entry data can be made using the IFSTRING command.

CROSS-REFERENCES:

RAPL COMMANDS: RETURN, ONERR, ONSIG

FORMATS:

Source: [Line#] GOTO <Line#>
Tokenized: [Line#] /075 <Line#>

DESCRIPTION:

An unconditional branch to another program line can be performed by this statement.

APPLICABLE MODES:

{P}

EXAMPLE:

1. Program will increment the value X and display the result to the terminal indefinitely.

```
050 ; PROGRAM MAIN
060 ;
090 ! X=0
100 ! X=X+1
110 TYPEI X
120 TYPE ' '/
130 GOTO 100
```

CROSS-REFERENCES:

RAPL COMMANDS: IF, IFSIG, IFSTART, IFPOWER all also redirect program flow but conditionally.

GRIP

FORMATS:

Source: [Line#] GRIP <distance>
Tokenized: [Line#] /039 <distance>

DESCRIPTION:

The GRIP command will open or close the gripper fingers. When the argument <distance> is less than the current finger position the fingers will close. When the argument <distance> is larger than the current finger position the fingers will open. The argument <distance> reflects the distance between the fingers and has a range from 0 to 2 inches.

The force applied to the object with the GRIP command is 100% always, compared to the CLOSE or OPEN commands which control gripper force but not position.

WARNINGS:

1. REQUIRES THE SRS-SGRIP/M1 OPTION
2. RAPL version 4.10 and higher have separated the gripper position and force modes. The OPEN and CLOSE commands are now active in the servo gripper mode, as they are used to specify the gripper force mode. This command is incompatible with earlier versions of RAPL and some programming change will be required for proper operation.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: OPEN, CLOSE, @@GTYPE

HALT

FORMATS:

Source: [Line#] HALT [ON <[-]Input#>]
Tokenized: [Line#] /154 [ON <[-]Input#>]

DESCRIPTION:

The HALT command will stop any current robot motion that is in progress. It is useful when having to recover from an ONSIG command that requires a robot motion to cease. In that mode, HALT would be the first command in the routine called by the ONSIG.

If the optional "HALT ON <Input>" is called, all robot motion will cease whenever the <Input#> is in the programmed state. Activation of this HOLD feature requires the ENABLE HOLD command to be issued.

APPLICABLE MODES:

{I}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: ABORT, PROCEED, ENABLE, DISABLE

OTHER CRS Plus Publications: SRS-APP/SAFETY, SRS-APP/PATH

HELP

FORMATS:

Source: **HELP**
Tokenized: /095

DESCRIPTION:

Turns the syntax building feature ON. In this mode, the programmer needs only to enter enough of the command to make it unique, then the syntax builder will finish typing the command identifier. Depending on the command, the syntax builder prompts the programmer to enter the balance of the command correctly.

The syntax builder can also be turned on by a <Ctrl-H> from a standard terminal. This control code will not work from ROBCOMM as it will be interpreted as the Back-Space key.

WARNINGS:

1. Make sure to type carefully with HELP enabled since any extra characters that are entered may be mistaken for data required later in the command line.

APPLICABLE MODES:

{I}, {M}

CROSS-REFERENCES:

RAPL COMMANDS: ENABLE, DISABLE, NOHELP

HERE

FORMATS:

Source: [Line#] HERE <LOC_NAME>
Tokenized: [Line#] /024 <LOC_NAME>

DESCRIPTION:

Defines a location as the current commanded robot position (contrast this with the ACTUAL command). The location can be either a precision point or cartesian location definition.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. For a cartesian location:

>>HERE POINT

2. For a precision point:

>>HERE #POINT

CROSS-REFERENCES:

RAPL COMMANDS: ACTUAL, POINT, LISTL

HOME

FORMATS:

Source: [Line#] HOME
Tokenized: [Line#] /020

DESCRIPTION:

When the robot is first turned on, the controller does not know the arm position relative to the world (as defined by the centre of the manipulator base). The operator must HOME the robot to provide the synchronization between controller and arm.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE

1. Enter manual mode:

>>MANUAL

Move all five joints with the teach pendant to marks on robot starting with joint 1 through to Joint 5.

Enter <HOME> at keyboard. The controller will ask you to confirm that you have moved the robot arm into the starting home range. If robot arm is in starting home range then enter <YES> at keyboard.

J>HOME LOCATED IN HOME BOUNDS?. Y<ES>

The controller will move each joint to the factory set home position and display the status on the terminal screen.

If an error occurs in any of the joints return to step (1) and repeat complete procedure. If a error occurs in three (3) consecutive attempts contact your local distributor or CRS Plus.

Visually inspect each homing marker to ensure that the robot is homed properly. If the arm did not home correctly return to step (1) and repeat complete procedure.

CROSS-REFERENCES:

RAPL COMMANDS: @CAL

FORMATS:

Source:

[Line#] IF <argm't> <EQ!NE!LT!GT!LE!GE> <argm't> THEN <Line#>

Tokenized:

[Line#] /072 <argm't> <EQ!NE!LT!GT!LE!GE> <argm't> THEN <Line#>

DESCRIPTION:

The expression is evaluated using one of the six tests listed below. If a true result is obtained, then control passes to the specified line number. Logical statements can be evaluated using the following expressions:

EQ equal to
 NE not equal to
 LT less than
 GT greater than
 LE less than or equal to
 GE greater than or equal to

The two arguments can be variables, or explicit values, or one of each.

WARNINGS:

1. It should be noted that since all values are stored as real numbers, it is possible to obtain confusing results when using the EQ operator, since the numbers may not be exactly equal due to the round-off errors associated with the storage of such elements.

APPLICABLE MODES:

{F}

EXAMPLE:

1. While the variable ROW is less than 5 this program will loop between line #90 and line #100.

```
90 ! ROW = ROW + 1
100 IF ROW LT 5 THEN 90
110 STOP THE TRAY IS FINISHED
```

CROSS-REFERENCES:

RAPL COMMANDS: GOTO, IFSIG, IFSTART, IFPOWER, IFSTRING

IFPOWER

FORMATS:

Source: [Line#] IFPOWER THEN <Line#>
Tokenized: [Line#] /155 THEN <line#>

DESCRIPTION:

Test the sense of the arm power switch, and if it is turned on, then branch to the line specified by the command.

APPLICABLE MODES:

{P}

CROSS-REFERENCES:

RAPL COMMANDS: GOTO, IF, IFSIG, IFSTART, IFSTRING

FORMATS:

Source: [Line#] IFSIG <[-]input#>[,...] THEN <Line#>
 Tokenized: [Line#] /073 <[-]input#>[,...] THEN <Line#>

DESCRIPTION:

The state of the input points are logically ANDed in this command. If any specified input does not match the required state, the test is false and program flow will continue to the next line. Should the test prove to be true, then program flow will proceed to the specified line number. Any number of inputs can be tested by this command, so long that the line length does not exceed the maximum line length permitted.

APPLICABLE MODES:

{P}

EXAMPLES:

1. If input#1 is at a high level, jump to Line 399, otherwise continue to next line.

```
100 IFSIG 1 THEN 399
```

2. If input#3 is at a low level, jump to Line #400, otherwise continue to next line.

```
101 IFSIG -3 THEN 400
```

3. If input#1 is at a high level AND input #2 is at a low level AND input #3 is at a high level AND input #4 is at a low level AND input #5 is at a high level AND input #6 is at a low level, jump to Line #101, otherwise continue to next line.

```
120 IFSIG 1,-2,3,-4,5,-6 THEN 101
```

CROSS-REFERENCES:

RAPL COMMANDS: GOTO, IF, IFSTART, IFSTART, IFPOWER

IFSTART

FORMATS:

Source: [Line#] IFSTART THEN <Line#>
Tokenized: [Line#] /086 THEN <Line#>

DESCRIPTION:

This is a specialized version of the IFSIG command. Here, the state of the AUTOSTART switch is examined. If it is in a high state, then program control is passed to the program number defined in the statement. If the input is low, then control passes to the next block in the program.

APPLICABLE MODES:

{P}

CROSS-REFERENCES:

RAPL COMMANDS: GOTO, IF, IFSIG, IFSTRING, IFPOWER

IFSTRING

FORMATS:

Source: [Line#] IFSTRING <&n;'TEXT'> EQ;NE <&n;'TEXT'> THEN <LINE#>
Tokenized: [Line#] /151 <&n;'TEXT'> EQ;NE <&n;'TEXT'> THEN <LINE#>

DESCRIPTION:

This command compares two strings using the "equal" condition only (EQ). If a true result is obtained, then control passes to the specified line number.

The two arguments can be string variables, or explicit values, or one of each. The line number can also be a variable.

APPLICABLE MODES:

{P}

CROSS-REFERENCES:

RAPL COMMANDS: GOTO, IF, IFSIG, IFSTART, IFPOWER

IGNORE

FORMATS:

Source: [Line#] IGNORE
Tokenized: [Line#] /087

DESCRIPTION:

This command will disable and clear the ONSIG state. In contrast, the "DISABLE ONSIG" command will disable but not clear the ONSIG state, permitting re-enabling (by "ENABLE ONSIG") without re-programming.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: ONSIG, DISABLE

FORMATS:

Source: [Line#] INPUT <VAR_NAME>|<&n>
 Tokenized: [Line#] /068 <VAR_NAME>|<&n>

DESCRIPTION:

This command permits the user to INPUT data into the program at run time. The specified variable or string specifier indicates the destination of the input information. The operator can enter another variable name in response to a variable type INPUT command (see EXAMPLE 1. below).

The current default device (1 or 0) is always the source of the data for this operation. After INPUT of a variable, a <CRLF> sequence will be echoed to the device. After a string INPUT, no <CRLF> is echoed.

APPLICABLE MODES:

{P}

EXAMPLES:

1. Check for program continuation using a variable INPUT:

```
2100 ! N = -1
2200 ! Y = 1
2300 TYPE 'Do you wish to proceed? (Y/N): '
2400 INPUT RESULT
2500 IF RESULT EQ Y THEN 3000
2600 IF RESULT EQ N THEN 2800
2700 GOTO 2300
2800 STOP Finished as requested.
3000 ; So continue...
```

2. Check for program continuation using a string INPUT:

```
2300 TYPE 'Do you wish to proceed? (Y/N): '
2400 INPUT &4
2450 TYPE ''/
2500 IFSTRING &4 EQ 'Y' THEN 3000
2600 IFSTRING &4 EQ 'N' THEN 2800
2700 GOTO 2300
2800 STOP Finished as requested.
3000 ; So continue...
```

CROSS-REFERENCES:

RAPL COMMANDS: DEVICE, !, all string commands

INVERT

FORMATS:

Source: [Line#] INVERT <ON;OFF>
Tokenized: [Line#] /170 <ON;OFF>

DESCRIPTION:

The SRS-M1A robot system can be mounted upside down to provide a clear table top work space. To facilitate programming the robot in this configuration, the Z axis of the robot coordinate system may be reversed. This reversal leaves the X and Y coordinate axes the same as before. Notice that while using the manual mode, and the JOINT command, the motions of the joints are opposite of what is expected. Also, when using the robot in the inverted position, the normal arm configuration is typically the elbow DOWN selection.

As an aid to programming, the Z base OFFSET parameter can be set to equal the distance between the robot base and the work surface. This way, the inverted coordinate system will relate directly to the work surface, which is usually more convenient.

WARNING:

Ensure that any locations taught with the INVERT ON condition are used in that fashion as well. Otherwise, unexpected results could occur. It is good programming practice to put the INVERT command at the start of any program using inverted locations.

APPLICABLE MODES

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: ELBOW, REACH

FORMATS:

Source: [Line#] JOG <dX>, <dY>, <dZ>
Tokenized: [Line#] /005 <dX>, <dY>, <dZ>

DESCRIPTION:

The JOG command permits the operator to move the robot by a specified cartesian increment in inches. The move is completed by a straight line motion with the gripper flange ending with the same orientation wherever possible.

This command is useful when positioning the robot at a desired position. Once close proximity has been achieved in the manual mode, the JOG command can be used to perform final positioning. The jog command will execute at the last setting of the SPEED command.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Move arm by +0.5 inches in the Y axis.

```
200 JOG 0,0.5,0
```

CROSS-REFERENCES:

RAPL COMMANDS: none

JOINT

FORMATS:

Source: [Line#] JOINT <Joint#>, <Degrees>
Tokenized: [Line#] /006 <Joint#>, <Degrees>

DESCRIPTION:

Will drive a selected joint by a specified displacement. The joints of the robot will move the specified distance in degrees. Optional extra axes will move a number of units from their current location. The desired units are selected by the @XPULSES and @XRATIO commands were executed.

Robot joints will move in a joint de-coupled fashion. The joint number describes which joint is moved by the following legend:

1. Waist
2. Shoulder
3. Elbow
4. Wrist Bend (pitch)
5. Wrist swivel (Roll)

The joint command will execute at a speed as specified in the last SPEED command. The sense of each joint can be seen in Figure 2-2.

WARNINGS:

1. Each robot joint has a limiting travel which should not be exceeded. This limit is checked prior to the execution of the JOINT command when the robot has been HOMED.
2. JOINT moves can be done before the robot has been homed, but limits will not be checked. Care must be taken with this command until the robot is homed.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Move Joint #1 (Base) by +45 degrees.

200 JOINT 1,45

CROSS-REFERENCES:

RAPL COMMANDS: @NOA, @ACCEL, @XPULSES, @XRATIO, @XLIMITS, @MAXVEL,
LOCK, UNLOCK

OTHER CRS Plus Publications: SRS-M1A/TECH, SRS-APP/EXTRA

FORMATS:

Source: [Line#] LIMP [Axis#]
Tokenized: [Line#] /007 [Axis#]

DESCRIPTION:

The LIMP command permits the operator to disengage all or some of the positional servos which maintain the robot position. The robot joints can then be moved freely.

The LIMP condition is terminated by the NOLIMP command. At termination of the LIMP, the commanded position of each axis will be their current position.

WARNINGS:

During limp motions, care must be exercised not to over-stress the robot arm when driving back through the transmission elements.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Limp all 8 robot axes.

200 LIMP

2. Limp axis #1 (Base).

200 LIMP 1

CROSS-REFERENCES:

RAPL COMMANDS: NOLIMP, ENABLE, DISABLE, @APC, @NAPC, ARM

LISTL

FORMATS:

Source: [Line#] LISTL [LOC_NAME][,0;1]
Tokenized: [Line#] /049 [LOC_NAME][,0;1]

DESCRIPTION:

This command will list a location stored in the robot memory. If no name is supplied, then a complete list of all locations along with their values will be provided in a tabular format.

The optional [1] argument will specify output to the printer port.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. List all locations to video display.
>>>LISTL
2. List location POINT to printer port.
>>>LISTL POINT,1

CROSS-REFERENCES:

RAPL COMMANDS: HERE, POINT, DLOC

FORMATS:

Source: [Line#] LISTP <PRG_NAME>[,0;1]
Tokenized: [Line#] /103 <PRG_NAME>[,0;1]

DESCRIPTION:

This command lists a program to the selected device. If no program name is specified, then the current program being edited will be listed.

The optional [1] argument will specify output to the printer port.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: EDIT

LISTV

FORMATS:

Source: [Line#] LISTV [,0;1]
Tokenized: [Line#] /069 [,0;1]

DESCRIPTION:

Lists all the variables stored in memory and displays their values.

The optional [1] argument will specify output to the printer.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. List all variables to video terminal.

```
>>LISTV
```

2. List all variables to printer.

```
>>LISTV ,1
```

CROSS-REFERENCES:

RAPL COMMANDS: !, DVAR, TYPEV, TYPEI, PRINTV, PRINTI

FORMATS:

Source: [Line#] LOCK <Joint#>[,...]
 Tokenized: [Line#] /009 <Joint#>[,...]

DESCRIPTION:

Any joints specified in the LOCK command will be excluded from any further motion commands. This will allow the controller to control two separate machines which must perform non-synchronized tasks.

This command is useful if uncoordinated motion is required as in the case where the robot controller is used to control the robot and a separate piece of equipment in a work cell. If at some time during the operation, this external piece of equipment had to perform an independent motion, it could be performed without affecting the timing of the robot motions. The programmer commands the equipment to move, LOCKs those axes of motion, then commands the robot to perform its task.

WARNING:

The locked status will not apply in MANUAL mode.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. A program which commands the robot to move to a position including a TRACK axis and then moves the arm only after locking the track axis:

```
100 TRACK Y
...
170 MOVE MACHINE1
175 ; Lock out the track from the next move commands
180 LOCK 6
200 APPROD PICK_1,2
210 MOVE PICK_1
...
```

CROSS-REFERENCES:

RAPL COMMANDS: UNLOCK, @NOA, @ACCEL, @XPULSES, @XRATIO, @XLIMITS,
 @MAXVEL

OTHER CRS Plus Publications: SRS-M1A/TECH, SRS-APP/EXTRA

MA

FORMATS:

Source: [Line#] MA <j1>,<j2>,<j3>,<j4>,<j5>
Tokenized: [Line#] /132 <j1>,<j2>,<j3>,<j4>,<j5>

DESCRIPTION:

This command will feed the robot controller with a set of absolute joint move commands. The robot will move all axes to the specified end locations in a joint interpolated motion. The joint angles are entered in radian units. The joint angles can be defined either by pre-defined variables, or by explicit values.

WARNINGS:

All five angles must be entered, each separated by a comma (preferred) or a space.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: MI

FORMATS:

Source: [Line#] MAGGRIP <% maximum>
Tokenized: [Line#] /042 <% maximum>

DESCRIPTION:

To specify the amount of force to be applied by the magnetic gripper. The programmer must enter a value between 0 and 100.

APPLICABLE MODES:

{I}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: @@GTYPE

MANUAL

FORMATS:

Source: [Line#] **MANUAL** [JOI ; CYL]
Tokenized: [Line#] /045 [JOI ; CYL]

DESCRIPTION:

The MANUAL mode allows the operator to move the robot using the 8 joint selector switches on the teach pendant, along with the gripper toggle switch and the speed selector knob.

During MANUAL operation, all non-motion commands remain active. Note that the system prompt changes from the usual >> after the MANUAL command is entered. The system console may operate slightly slower in the manual mode due to the extra scanning which is required for the teach pendant.

There are two modes of robot control in MANUAL. In the JOINT MANUAL mode, individual joints are controlled with each toggle switch on the teach pendant. In CYLINDRICAL, the motion of joints 2 and 3 are coordinated to provide a motion of the wrist that is radial, with constant elevation, or vertical (in the Z axis), with constant radial distance from the robot base. The base rotate and the two wrist axes are controlled the same as in JOINT mode. This mode cannot be used until the robot has been HOMEd.

The following table illustrates the functions of the 8 switches at different robot configurations:

SWITCH #	JOINT mode 5 - 6 axes	CYL mode 5 - 6 axes	JOINT mode 7 - 8 axes	CYL mode 7 - 8 axes
1	JOINT 1	JOINT 1	JOINT 1	JOINT 1
2	JOINT 2	RADIUS	JOINT 2	RADIUS
3	JOINT 3	VERTICAL	JOINT 3	VERTICAL
4	JOINT 4	JOINT 4	JOINT 4	JOINT 4
5	JOINT 5	JOINT 5	JOINT 5	JOINT 5
6	-	-	JOINT 6	JOINT 6
7	ALIGN	ALIGN	JOINT 7	JOINT 7
8	LIMP/NOL.	LIMP/NOL.	JOINT 8	JOINT 8

APPLICABLE MODES:

{I}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: NOMANUAL, ENABLE, DISABLE

FORMATS:

Source: [Line#] MEMRD [ITEM],[ADDRESS],[VAR_NAME]
 Tokenized: [Line#] /104 [ITEM],[ADDRESS],[VAR_NAME]

DESCRIPTION:

This command will read the contents of memory that is specified at the physical address location. This address can be supplied as a constant, or as a variable name. The address must be within the physical address limits of the 8086 processor (0-1048575) and in decimal notation.

The contents of the memory location is returned and loaded into the variable name that is provided. The type of memory access is specified by the [ITEM] argument, which is a string with one of the following values:

BYTE (0 to 255)
 WORD (0 to 65535)
 INTEGER (-32768 to 32767)
 REAL
 DINTEGER (double integer)

APPLICABLE MODES:

{I},{P},{M}

EXAMPLE:

1. In an program entered by the ONERR command, determine the error type knowing the address of the "Alarm Number":

```
1000 ; PROGRAM ERR_RECV:
1100 DISABLE ARM
1200 MEMRD BYTE,7603,ERR_NUM
1300 TYPE '*** Error encountered - '
1400 TYPEI ERR_NUM
1500 TYPE ' ***'/
1600 STOP
```

CROSS-REFERENCES:

RAPL COMMANDS: MEMWR

OTHER CRS Plus Publications: SRS-M1A/TECH

MEMWR

FORMATS:

Source: [Line#] MEMWR [ITEM],[ADDRESS],[VAR_NAME;CONSTANT]
Tokenized: [Line#] /118 [ITEM],[ADDRESS],[VAR_NAME;CONSTANT]

DESCRIPTION:

This command will write the constant, or the contents of the supplied variable to the physical address location. This address can be supplied as a constant, or as a variable name. The address must be within the physical address limits of the 8086 processor (0-1048575) and in decimal notation.

BYTE (0 to 255)
WORD (0 to 65535)
INTEGER (-32768 to 32767)
REAL
DINTEGER (double integer)

WARNING:

1. The programmer can easily corrupt the robot operational parameters with this command. It is therefore suggested that the programmer contact CRS for technical assistance.

APPLICABLE MODES:

{I},{P},{M}

CROSS-REFERENCES:

RAPL COMMANDS: MEMRD

OTHER CRS Plus Publications: SRS-M1A/TECH

FORMATS:

Source: [Line#] MI <j1>,<j2>,<j3>,<j4>,<j5>
Tokenized: [Line#] /131 <j1>,<j2>,<j3>,<j4>,<j5>

DESCRIPTION:

This command will feed the robot controller with a set of incremental joint move commands. The robot will move all axes by the specified amounts in a joint interpolated motion. The joint increments are entered in radian units. The joint increments can be defined either by predefined variables, or by explicit values.

WARNINGS:

All five angles must be entered, each separated by a comma (preferred) or a space.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: MA

MOTOR

FORMATS:

Source: [Line#] MOTOR <axis#>, <pulses>
Tokenized: [Line#] /011 <axis#>, <pulses>

DESCRIPTION:

Will drive the selected motor by the specified number of pulses. Consult the technical reference manual for the description of the number of encoder pulses per motor turn. Note that this command controls individual motors, not axes.

If two joints are coupled, then a MOTOR command will move both joints. Also remember that the motor coordinates may be reversed from what you may expect, so treat this command with caution.

WARNINGS:

1. The joint limit checking function does not limit the travel of the robot during this mode.
2. The FINISH command is not implicit in a MOTOR command. In the following sequence, the command at 100 will not finish before the command at 200 starts. However, line 300 will not start until 200 is finished, nor will 500 start before 400 is done.

```
100 MOTOR 2,2500  
200 MOTOR 3,-7500  
300 MOTOR 3,5000  
400 MOTOR 1,2000  
500 MOVE A
```

APPLICABLE MODES:

{I}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: none

OTHER CRS Plus Publications: SRS-M1A/TECH

FORMATS:

Source: [Line#] MOVE <LOC_NAME>[,S]
Tokenized: [Line#] /012 <LOC_NAME>[,S]

DESCRIPTION:

Will move the robot to the specified location. The location can be either a cartesian coordinate, or a precision point.

A MOVE command will utilize a joint interpolated motion, where the individual robot joints are commanded in such a way that they start and stop at the same time.

The straight line [,S] argument will instruct the robot to move so that the tool tip follows a straight line path to the destination. The MOVE command proceeds at the previously defined SPEED setting.

MOVE will normally coordinate only the five axes of the robot. It will coordinate the extra axes under the following conditions:

1. More than the 5 axes are in the system and a precision point location is specified.
2. A TRACK or GANTRY command has been issued prior to the move. In this case all locations, cartesian and precision point include extra axis coordinates.

APPLICABLE MODES:

{I}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: APPRO, DEPART, FINISH, HERE

NEW

FORMATS:

Source: **NEW**
Tokenized: /021

DESCRIPTION:

This command will clear the current user memory without changing the memory partitioning. Upon execution of the command, the user will be asked if he is sure. All programs, locations and variables will be erased if a Yes response is given.

APPLICABLE MODES:

{I}, {M}

EXAMPLE:

1. Clear all memory, Enter:

```
>>NEW<cr>
```

response:

```
are you sure? Y<ES> <cr>
```

CROSS-REFERENCES:

RAPL COMMANDS: ALLOC, FREE, DIR, LISTL, LISTV

FORMATS:

Source: NEXT [PRG_NAME],[Line#]
Tokenized: /105 [PRG_NAME],[Line#]

DESCRIPTION:

Single stepping through a program can often be a useful tool in debugging programs. Once a program and line count has been specified with a NEXT statement, they need not be entered again, since the program line number is automatically incremented to the next line. The NEXT command can also be used after a <Ctrl-A> program abort. The PROCEED command can be used to continue program execution after a NEXT command.

NEXT can be used with TRACE enabled to provide single-stepping with a display of the current program status.

APPLICABLE MODES:

{I},{M}

EXAMPLE:

1. To start a program (MAIN in this example) at a point other than the first line (line 1350 in this example) type:

```
>>NEXT MAIN,1350<cr>
```

After that program line has been executed (check using STATUS), type:

```
>>PROCEED
```

CROSS-REFERENCES:

RAPL COMMANDS: RUN, PROCEED, TRACE, NOTRACE

NOFLASH

FORMATS:

Source: [Line#] NOFLASH
Tokenized: [Line#] /088

DESCRIPTION:

This command will turn off the FLASH command, and will reinstate the error condition output to the READY light.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: FLASH, ENABLE, DISABLE

NOHELP

FORMATS:

Source: NOHELP
Tokenized: /096

DESCRIPTION:

Turns the syntax building feature OFF. This makes the command entry a completely manual affair. When controlling the robot through a host computer interface, the syntax builder should be turned off.

APPLICABLE MODES:

{I}, {M}

CROSS-REFERENCES:

RAPL COMMANDS: HELP, ENABLE, DISABLE

NOLIMP

FORMATS:

Source: [Line#] NOLIMP [Axis#]
Tokenized: [Line#] /008 [Axis#]

DESCRIPTION:

Re-establishes the closed loop servo control after a LIMP command has been issued. (Refer to description of LIMP command.)

When the robot arm power is off, the robot is effectively in the limp mode. This is so that whenever the arm power is turned back on, the robot will assume this current position under closed loop control.

APPLICABLE MODES:

{I}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: LIMP, ENABLE, DISABLE

NOMANUAL

FORMATS:

Source: [Line#] NOMANUAL
Tokenized: [Line#] /046

DESCRIPTION:

This function will cancel a previously assigned MANUAL command. This will return the system prompt to the >> symbol. (Refer to description of MANUAL command.) Commanding a motion through the terminal will automatically execute a NOMANUAL command.

APPLICABLE MODES:

{M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: MANUAL, ENABLE, DISABLE

NOTEACH

FORMATS:

Source: [Line#] **NOTEACH**
Tokenized: [Line#] /043

DESCRIPTION:

This statement will deactivate the TEACH mode. (Refer to description of TEACH command.)

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: TEACH, ENABLE, DISABLE, CTPATH, WITH

OTHER CRS Plus Publications: SRS-APP/PATH

NOTRACK

FORMATS:

Source: [Line#] NOTRACK
Tokenized: [Line#] /080

DESCRIPTION:

This will disable the TRACE mode. (Refer to description of TRACE command.)

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: TRACE, ENABLE, DISABLE

OCOMP

FORMATS:

Source: [Line#] OCOMP <LOC_NAME>, <VAR_NAME>
Tokenized: [Line#] /064 <LOC_NAME>, <VAR_NAME>

DESCRIPTION:

The O component (orientation or yaw angle) of "LOC_NAME" is extracted and stored in "VAR_NAME". The angle is stored in degrees.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: COMP, XCOMP, YCOMP, ZCOMP, ACOMP, TCOMP

FORMATS:

Source: [Line#] OFFSET <dX>, <dY>, <dZ>, <dO>
 Tokenized: [Line#] /047 <dX>, <dY>, <dZ>, <dO>

DESCRIPTION:

The centre of the base of the robot defines the cartesian point (0,0,0) in the default condition. The base coordinates can be re-defined using the OFFSET statement. Each of the three cartesian coordinates can be offset, along with the YAW angle. All further robot commands which reference the world coordinate system will be made relative to this new coordinate system.

OFFSET can be used to set the "Z=0" plane at the work surface in the case where the robot is suspended from a gantry frame and the INVERT command has been issued.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Two arrays of components to be picked are +2" in the X dimension, +2" in the Y dimension, and at a 45 degree to each other. A program to pick parts from both fixtures would be:

```
....
100 GOSUB GET_PRTS
150 OFFSET 2,2,0,45
200 GOSUB GET_PRTS
....
```

2. A robot is inverted on a gantry frame 28 inches above the work surface. To set the world coordinate to a Z value of 0 at the table the following is recommended:

```
100 INVERT ON
200 OFFSET 0,0,-28,0
....
```

CROSS-REFERENCES:

RAPL COMMANDS: MOVE, SHIFT, INVERT

ONERR

FORMATS:

Source: [Line#] ONERR <PRG_NAME>
Tokenized: [Line#] /143 <PRG_NAME>

DESCRIPTION:

Normally when an error occurs during the execution of a program, the robot is halted and the program terminates. By using the ONERR command, an error during the program will cause program control to jump to the named program (PRG_NAME). This is useful in the case of an arm power error due to the use of the "ARM OFF" command in a program for instance.

APPLICABLE MODES:

{P}

EXAMPLE:

1. If an error occurs during a program, it is useful to determine what the error is. If no terminal is in the system, the message will not be displayed, and once the arm power is turned off, the last error STATUS line will always show error "040 - ARM POWER". To trap the error, the following may be used:

```
100 ; In MAIN:
200 ONERR GET_ERR
300 ....

1000 ; PROGRAM GET_ERR:
1100 DISABLE ARM
1200 MEMRD BYTE,7603,ERR_NUM
1300 TYPE '** Error encountered -'
1400 TYPEI ERR_NUM
1500 TYPE ' ***' /
1600 RETURN
```

CROSS-REFERENCES:

RAPL COMMANDS: GOSUB, ONSIG, @APC, @NAPC, ENABLE, DISABLE, MEMRD

FORMATS:

Source: [Line#] ONPOWER
Tokenized: [Line#] /156

DESCRIPTION:

Test the sense of the arm power switch, and hold until the arm power is turned on. When the arm power is turned on, proceed to the next program line. Unlike the ONSTART command, this command does not look for a transition of the arm power state. Thus if the arm power is on when this command is entered, control will pass to the next line.

This command is useful for auto-start routines (AUTO_ST) where the program must wait for the operator to turn on the arm power.

APPLICABLE MODES:

{I}, {P}, {M}

CROSS-REFERENCES:

RAPL COMMANDS: none

OTHER CRS Plus Publications: SRS-M1A/TECH (Chapter 9)

ON SIG

FORMATS:

Source: [Line#] ON SIG <[-]Port#>, <Prg_Name>
Tokenized: [Line#] /090 <[-]Port#>, <Prg_Name>

DESCRIPTION:

This command will set up RAPL to react to an input immediately by branching to the specified sub-program.

Once the ON SIG condition has been activated, the specified input will be sensed every approximately 40 milliseconds. When the input matches the specified condition, the controller will execute the subroutine that is specified in the command line.

The input used can be any user input. If the input is a pulse, the pulse must be at least 100 milliseconds in length to be sure it is captured reliably by the system.

APPLICABLE MODES:

{P}

CROSS-REFERENCES:

RAPL COMMANDS: IGNORE, ENABLE, DISABLE, GOSUB, RETURN

OTHER CRS Plus Publications: SRS-APP/SAFETY, SRS-APP/PATH

FORMATS:

Source: [Line#] ONSTART
Tokenized: [Line#] /092

DESCRIPTION:

This is a specialized version of the WAIT command. In this command, the state of the AUTOSTART switch is scanned. When the state of the input changes from low to high, control is passed to the next line of the program. Note that this command waits for the transition not just for the state to be high. Thus if the switch is depressed when the command is executed, it must be released and pressed again for program flow to continue. This prevents one trip of the switch from passing through multiple ONSTART commands in the same program.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: ONPOWER, IFSTART

OPEN

FORMATS:

Source: [line#] OPEN [% FORCE]
Tokenized: [line#] /040 [% FORCE]

DESCRIPTION:

With a pneumatic gripper installed, this command will open the gripper. When in Manual mode the gripper can be opened by use of the gripper switch on teach pendant.

When the servo gripper is installed, then the OPEN command serves to open the gripper at a specified force, as a percentage of full force.

APPLICABLE MODES:

{I}, {P}

EXAMPLE:

1. To open an air gripper:
 >>OPEN
2. To open the SERVO gripper at a torque of 60%:
 >>OPEN 60

CROSS-REFERENCES:

RAPL COMMANDS: CLOSE, GRIP, @@GTYPE

OTHER CRS Plus Publications: SRS-APP/DBLGRIP

FORMATS:

Source: [Line#] OUTPUT <[-]output#>[,...]
Tokenized: [Line#] /089 <[-]output#>[,...]

DESCRIPTION:

Any number of output points can be switched on or off by this command. The acceptable output number ranges from 1 to 40 with the extended option, or from 1 to 16 standard. A sign before each output number describes the requested state of the output. A negative sign before an output channel number will cause that output to go low. The positive sign is unnecessary.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLES:

1. Sets output #1 to a high level.
100 OUTPUT 1
2. Sets output #2 to a low level.
110 OUTPUT -2
3. Sets outputs #1, #3 and #5 to a high level and #2 and #4 to a low level.
120 OUTPUT 1,-2,3,-4,5

CROSS-REFERENCES:

RAPL COMMANDS: TRIGGER, ENABLE, DISABLE

PASSWORD

FORMATS:

Source: PASSWORD <password>
Tokenized: /141 <password>

DESCRIPTION:

Enters supervisory mode. Entry of the correct value will permit access to monitor level commands. In addition, the monitor commands will be displayed in the "?" command. The supervisory mode may be disabled by entering an incorrect password.

WARNING:

After entry of the command, the terminal echo is disabled so that a private password entry can be made. If the operator aborts the PASSWORD command using the ABORT pushbutton, or a control C character from the keyboard, the terminal echo will remain disabled until re-enabled by the control E character or by the use of the CONFIG command.

APPLICABLE MODES:

{I}, {M}

CROSS-REFERENCES:

RAPL COMMANDS: ?

OTHER CRS Plus Publications: SRS-M1A/TECH

FORMATS:

Source: [Line#] PASTE <STRING1;'Text'>,<STRING2>,<CHAR_INDEX>
Tokenized: [Line#] /146 <STRING1;'Text'>,<STRING2>,<CHAR_INDEX>

DESCRIPTION:

This command will take the first string, and will insert, or paste it into the second string at the character index provided. Notice that this command requires the string symbol and number, not just the string index.

APPLICABLE MODES:

{I},{P},{M}

EXAMPLE:

1. Assign values to strings 1 and 2. Insert string 2 into string 1 starting at character index 8.

```
>>! &1 = 'INSERT><THIS'  
>>! &2 = 'SOURCE'  
>>PASTE &2,&1,8  
>>TYPE &1  
INSERT>SOURCE<THIS  
>>
```

CROSS-REFERENCES:

RAPL COMMANDS: CUT, STRPOS, !, ENCODE, DECODE, IFSTRING are other string related commands.

PAUSE

FORMATS:

Source: [Line#] PAUSE [message string]
Tokenized: [Line#] /076 [message string]

DESCRIPTION:

The PAUSE statement will halt program flow, and display the desired message on the terminal. A PROCEED command from the operator will then resume program control.

APPLICABLE MODES:

{P}

EXAMPLE:

1. In this example, the program will halt after the MOVE command and the message 'INSERT NEW PART, THEN TYPE "PROCEED"' will appear on the screen of the terminal. When the operator enters 'PROCEED', the program will continue.

```
920 MOVE AWAY  
930 PAUSE INSERT NEW PART, THEN TYPE "PROCEED"  
940 APPRO PART,3
```

CROSS-REFERENCES:

RAPL COMMANDS: STOP, RUN, PROCEED

FORMATS:

Source: POINT <LOC_NAME>
 Tokenized: /050 <LOC_NAME>

DESCRIPTION:

Defines a location, and allows its current value to be changed from the terminal before storage. A cartesian location requires six values, the X,Y,Z coordinates and Yaw, Pitch and Roll angles of the tool. The orientation angles are provided in degrees. A precision point value requires pulse encoder unit inputs for each axis of motion.

APPLICABLE MODES:

{I}, {M}

EXAMPLES:

1. For a cartesian location (after a GANTRY command has been issued):

>>POINT Enter location name: PART7

NAME	X/TRACKX	Y/TRACKY	Z	Yaw	Pitch	Roll
PART7	+000.0000	+000.0000	+000.0000	+000.0000	+000.0000	+000.0000
	+000.0000	+000.0000				

CHANGE? Y

X,Y,Z,Y,P,R,Gantry-X,Gantry-Y :
 12,15.75,10.875,0,90,0,0,2

2. For a precision point:

>>POINT Enter location name: #AA

NAME	AX#1/6	AX#2/7	AX#3/8	AX#4	AX#5
#AA	+0000001200	+0000001000	+0000002000	+0000000000	+0000000212

CHANGE? Y

(m1),(m2),(m3),(m4),(m5),(m6),(m7),(m8) :
 1200,1000,-2000,0,212

>>

CROSS-REFERENCES:

RAPL COMMANDS: LISTL, HERE, ACTUAL

PRINT

FORMATS:

Source: [Line#] PRINT <'text'!&STR_NUM>[argument]
Tokenized: [Line#] /055 <'text'>[argument]

DESCRIPTION:

PRINT commands are automatically routed to the default printer port, which is device #1. The optional arguments provide additional display duties. The following are valid arguments:

- / Include a carriage return/line feed combination after the line
- * Include only a carriage return after the text.
- B Output the BELL character to the display.
- F Output a form feed to the device.

WARNINGS:

Attempting to use this command with device 1 configured as an ACI port will cause a 'Reserved I/O' error #59.

APPLICABLE MODES

{I},{M},{P}

EXAMPLE:

1. Prints "TEST" on printer.
>>PRINT 'TEST'
2. Prints "TEST" on printer with a carriage return and line feed.
>>PRINT 'TEST' /
3. Prints "TEST" on printer with a carriage return.
>>PRINT 'TEST' *
4. Print a string on the printer
>>PRINT &1

CROSS-REFERENCES:

RAPL COMMANDS: TYPE, PRINTI, PRINTV

FORMATS:

Source: [Line#] PRINTI <VAR_NAME>[,...]
Tokenized: [Line#] /056 <VAR_NAME>[,...]

DESCRIPTION:

PRINTI commands are automatically routed to the default printer port, which is device #1.

Often it is useful to display variables in an integer format, since a variable may only represent an integer value. A counter which must be displayed each loop looks far more informative when shown as a '+1' instead of '+1.0000' as a real variable would be displayed.

If the value does not fit within the integer limits (+/- 32766) then the display will be shown in real format.

Each typed field will be separated by a single space. No carriage return or line feed control is included in this command, since it is often useful to display more than one value on a display line.

WARNINGS:

Attempting to use this command with device 1 configured as an ACI port will cause a 'Reserved I/O' error #59.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: TYPEI, PRINT, PRINTV

PRINTV

FORMATS:

Source: [Line#] PRINTV <VAR_NAME>[,...]
Tokenized: [Line#] /057 <VAR_NAME>[,...]

DESCRIPTION:

PRINTV commands are automatically routed to the default printer port, which is device #1.

Each typed field will be separated by a single space. No carriage return or line feed control is included in this command, since it is often useful to display more than one value on a display line.

WARNINGS:

Attempting to use this command with device 1 configured as an ACI port will cause a 'Reserved I/O' error #59.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Prints variable value RADIUS to printer.

```
>>PRINTV RADIUS<cr>  
+002.7500
```

2. Prints variable values RADIUS and CIRCUM to printer.

```
>>PRINTV RADIUS, +002.7500 CIRCUM<cr>  
+008.6393  
>>
```

CROSS-REFERENCES:

RAPL COMMANDS: TYPEV, PRINT, PRINTI

FORMATS:

Source: [Line#] **PROCEED**
Tokenized: [Line#] /106

DESCRIPTION:

The PROCEED command has two distinct functions: it can be used in the immediate mode in order to re-start a program that has terminated prematurely, or it can be used in a program in order to re-start a continuous path that was interrupted, typically by an ONSIG condition.

Following the detection of a PAUSE statement in the program, a PROCEED command will cause the program to continue starting at the next program line. PROCEED may also be used after a 'soft' abort: that is, typing <Ctrl-A>. It should not be used after a hard <ABORT> (<Ctrl-C> or ABORT pushbutton on pendant) as this could cause unexpected results.

PROCEED can also be used after the NEXT command in some instances during program debugging, however, if the NEXT command was used to position the program pointer to within a subroutine, the RETURN statement could cause an error.

When PROCEED is used to continue a path, the command includes an implicit joint interpolated move back to the location where the path was interrupted.

WARNINGS:

1. The PROCEED command will resume a continuous path only if no other path is programmed. This includes any straight line move, in addition to CPATH or GOPATH commands.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLES:

1. After a program has terminated by a <ctrl-A> program pause or with the PAUSE command, or the program is being stepped through using the NEXT command, normal program operation can be resumed by entering:

>>PROCEED<cr>

Continued....

PROCEED (Cont)

2. If a continuous path move is interrupted by a HALT command within a program, the PROCEED command will cause it to resume from where it stopped.

```
PROGRAM: MAIN
10 ONSIG 1,ERR
20 CPATH A,B,C,D,E,F,G,H,J,K,L,M,N
30 FINISH
40 STOP
```

```
PROGRAM: ERR
10 HALT
20 DEPART 2
30 MOVE SAFE
40 WAIT -1
45 ; Move back to the path and continue:
50 PROCEED
60 RETURN -1
```

The ERR sub-program will be triggered when input #1 goes high. The robot will immediately stop, depart from the path, and move to a safe location. There it will wait for the signal that everything is clear, do a joint interpolated move back to the location at which it was interrupted and resume the path.

When programming this type of sequence, ensure that the location SAFE is accessible from any point along the path. If this is not the case, a collision may result during the retreat or the joint interpolated move back to the path. Also ensure that no path-type moves are executed before the PROCEED command or the path pointer will be cleared. This will prohibit the PROCEED from functioning as desired. Note that any straight move is a path-type command.

CROSS-REFERENCES:

RAPL COMMANDS: PAUSE, RETRY, CPATH, GOPATH

OTHER CRS Plus Publications: SRS-APP/PATH

FORMATS:

Source: [Line#] REACH <FORWARD;BACK>
Tokenized: [Line#] /171 <FORWARD;BACK>

DESCRIPTION:

When the SRS-M1A arm is mounted in the centre of the work area and working down on the mounting surface, it is in the REACH FORWARD mode. The SRS-M1A can, however, access a portion of its working volume "over its head". In some cases, a single location in space can be reached in both modes. Optimum cycle time may be achieved by using the REACH BACK command. For the M1A, the REACH BACK configuration must be accompanied by an ELBOW DOWN command since when the arm is reaching back, the elbow joint is pointing down. Refer to the description of the ELBOW command.

CROSS-REFERENCES:

RAPL COMMANDS: ELBOW, INVERT

READY

FORMATS:

Source: [Line#] READY
Tokenized: [Line#] /015

DESCRIPTION:

Moves arm to "READY" position shown in Figure 4-1 below.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Move arm to "READY" position shown in Figure 4-1.

>>READY

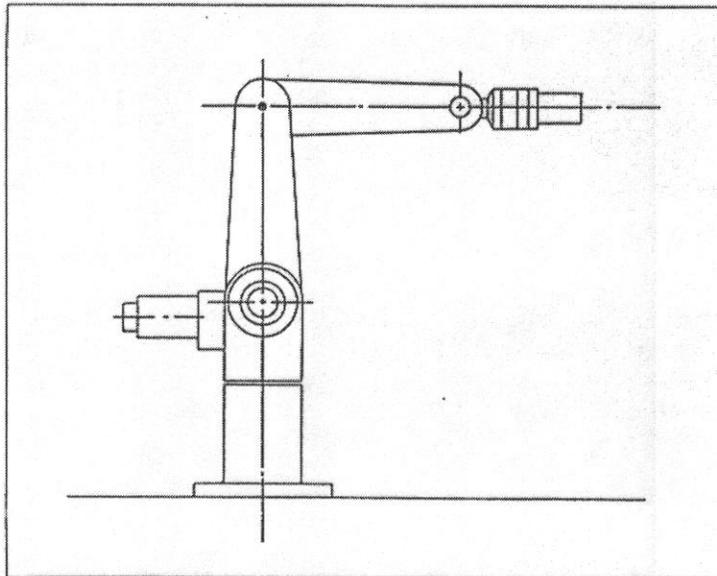


FIGURE 4-1 Arm "READY" Position

CROSS-REFERENCES:

RAPL COMMANDS: HOME, @@CAL, XREADY

OTHER CRS Plus Publications: SRS-M1A/TECH

FORMATS:

Source: RENAME <old PRG_NAME>, <new PRG_NAME>
Tokenized: /107 <old PRG_NAME>, <new PRG_NAME>

DESCRIPTION:

This command will change the name of a program already stored in memory. This function could be used when loading a new program through ROBCOMM. If an existing program has the same name, an error will result. Renaming the old program beforehand will eliminate this problem.

APPLICABLE MODES:

{I}, {M}

CROSS-REFERENCES:

RAPL COMMANDS: COPY, EDIT, DELETE

RETRY

FORMATS:

Source: **RETRY**
Tokenized: /108

DESCRIPTION:

If an error causes premature program termination, **RETRY** can be used to try the erroneous line again after the error has been corrected. The program will then execute as though it had not been interrupted, starting at the bad line. The program loop count status will be maintained.

WARNINGS:

1. If the robot was in motion when the error occurred, the error will have caused the arm to halt somewhere in its path. The **RETRY** command will not continue the motion but will pick up program execution at the line that caused the error. motion commands following this error may cause unexpected results. Caution with this command is advised at all times.

APPLICABLE MODES:

{I}, {M}

CROSS-REFERENCES:

RAPL COMMANDS: **PROCEED, RUN**

RETURN

FORMATS:

Source: [Line#] RETURN [skip #]
Tokenized: [Line#] /077 [skip #]

DESCRIPTION:

This statement is used to return program control to the calling program. By using the optional line skip counter, the return can include skipping over one or more lines when returning to the calling program.

Up to 255 lines can be skipped with this function. Omitting the skip count is the same as programming a skip count of 0. That is, the next line in the main program immediately following the call to the sub-program will be executed next.

Note that the skip number refers to numbers of lines and not to the numeric value of the line numbers in the calling program.

There is a special case of the skip number which is useful when calling a routine from an ONSIG condition. A value of -1 can be used in this case to ensure a return to the exact line which was being executed at the time of the ONSIG condition being satisfied. For instance if an onsig is anticipated in a motion command, follow this command with an explicit FINISH command. If the ONSIG occurs, a RETURN -1 at the end of the subroutine will return to the FINISH command.

APPLICABLE MODES:

{P}

Continued....

RETURN (Cont)

EXAMPLE:

1. Main is a routine called from the AUTO_Start program. At the end of the day's production, switch 2 is set, signalling a stop. This condition is checked in WAIT and returned using the line skip feature. In MAIN, when the stop signal has been received, flow returns to the AUTO_ST routine.

```
100 ; MAIN PROGRAM
110 GOSUB WAIT
120 RETURN
122 ; Task starts here:
130 MOVE SAFE
140 ...
...
500 GOTO 110

100 ; PROGRAM WAIT
110 IFSIG 1 THEN 150
120 IFSIG 2 THEN 140
130 GOTO 110
140 RETURN
150 RETURN 1
```

2. To stop the robot when a light curtain is broken, and re-start it when restored, the ONSIG command is used with a "RETURN -1" statement to terminate the called routine (HALT):

```
400 ....
410 ; Move safely:
420 ONSIG -1,HALT
430 APPRO PICK1,2
440 MOVE PICK1
450 FINISH
460 ...

PROGRAM HALT:
1000 SET DEST = ENDPNT
1100 HALT
1200 WAIT 1
1300 MOVE DEST
1400 RETURN -1
$
```

In this case, program flow cannot pass to the next line until after the commanded motion is complete. If signal 1 goes low, control passes to HALT which determines the motion destination (line 1000), stops the robot, waits for the signal 1 to go high, and repeats the move command. The -1 skip on the return will put the program pointer back to the correct line in the calling program.

CROSS-REFERENCES:

RAPL COMMANDS: GOSUB, ONSIG, ONERR

OTHER CRS Plus Publications: SRS-APP/PATH

FORMATS:

Source: [Line#] RUN [<PRG_NAME>][,loop count|F]
Tokenized: [Line#] /109 [<PRG_NAME>][,loop count]

DESCRIPTION:

This command will execute a program from the robot memory. If no program name is given, then the program that was last executing is run. If that program is a subroutine normally called by a main program, an error will likely result.

Program execution will start with the first statement in the program.

A program loop counter can be included in the statement. If no loop counter is supplied, then it is assumed to be 1. The maximum loop count is 65535, optionally entering an 'F' (Forever) as the loop count will result in infinite looping.

WARNINGS:

The RUN command may be used in a program with the following limitations:

- 1) The command must not be executed as many as 50 times without causing a stack error in the system computer and causing a shutdown.
- 2) Returning to the interactive mode may cause an unexpected error: probably error '012 COMMAND ERROR'. The system will also have disabled the HELP mode when it returns.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: RETRY, PROCEED, PAUSE, STOP

OTHER CRS Plus Publications: SRS-M1A/TUTORIAL

SERIAL

FORMATS:

Source: [Line#] SERIAL
Tokenized: [Line#] /111

DESCRIPTION:

This command will display the current setup of the two serial ports. It can be used before a CONFIG command so that the programmer is sure of the current settings of the port.

The display will always be routed to the default device.

APPLICABLE MODES:

{I}

EXAMPLE:

1. Enter:

>>SERIAL<cr>

Response:

Attribute	0	1
Baud Rate	9600	2400
Parity	Even	None
Data Bits	8	8
Stop Bits	2	2
Xon/Xoff	On	On
RTS/CTS	Off	Off
Echo	Off	Off

CROSS-REFERENCES:

RAPL COMMANDS: CONFIG

OTHER CRS Plus Publications: SRS-M1A/TECH

FORMATS:

Source: [Line#] SET <LOC_NAME> = <LOC_NAME>
 Tokenized: [Line#] /051 <LOC_NAME> = <LOC_NAME>

DESCRIPTION:

The SET function will equate one location to another. This command is useful when a location must be modified, but its original value must also be maintained.

The SET command can be used to assign user defined locations to reserved internal robot locations. These locations refer to the current commanded position, the actual position, and the current end point of the robot. These specific locations are accessible through reserved location names. The reserved locations can be determined in cartesian or precision point values. These reserved names are:

ACTUAL, #ACTUAL - Actual robot position registers
 POSCOM, #POSCOM - Commanded position registers
 ENDPNT, #ENDPNT - End point or destination registers for any motion. For PATH motion, this contains the last knot of the path.

Never attempt to write to these registers. If any operations are to be performed on this data, it is best assigned to a working location that is defined by the user using the SET command.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. A program used to halt any joint interpolated motion is below. Typically this would be used with a light curtain input (denoted LC here). It uses the SET command to determine where the robot is headed when interrupted:

```
100 SET DEST = ENDPNT
200 HALT
300 WAIT -LC
400 MOVE DEST
500 RETURN -1
```

CROSS-REFERENCES:

RAPL COMMANDS: HERE, POINT, SHIFT, SHIFTA

OTHER CRS Plus Publications: SRS-APP/PATH, SRS-APP/PALLET

SHIFT

FORMATS:

Source: [Line#] SHIFT <LOC_NAME> BY <dX>, <dY>, <dZ>
Tokenized: [Line#] /052 <LOC_NAME> BY <dX>, <dY>, <dZ>

DESCRIPTION:

In this command, the X, Y and Z values of the location are incremented by the specified amounts. A precision point location cannot be changed using SHIFT.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. SHIFT point 'P' by 1,1,0
120 SHIFT P BY 1,1,0
2. SHIFT point 'A' by .002,1.2,-0.5
>>! DX = 0.002<cr>
>>! DY = 1.2<cr>
>>! DZ = -.5<cr>
...
160 SHIFT A BY DX,DY,DZ

CROSS-REFERENCES:

RAPL COMMANDS: SET, SHIFTA, HERE

OTHER CRS Plus Publications: SRS-APP/PALLET

FORMATS:

Source:

[Line#] SHIFTA <LOC_NAME> BY <dX>, <dY>, <dZ>, <dO>, <dA>, <dT>[, <DTrkX>, <DTrkY>]

Tokenized:

[Line#] /053 <LOC_NAME> BY <dX>, <dY>, <dZ>, <dO>, <dA>, <dT>[, <DTrkX>, <DTrkY>]

DESCRIPTION:

Similar to the SHIFT function, the SHIFTA function permits all components of a cartesian location to be changed. A precision point location can also be changed in this fashion.

For cartesian locations, the X, Y, and Z coordinates are in inches, while the tool angles are in degrees. For precision points, motor pulses are used.

WARNINGS:

All coordinates must be entered. For precision points, one coordinate per axis. For cartesian locations, all six coordinates. If a TRACK or GANTRY has been specified, the corresponding axis values for these must also be entered in the programming units for the axis in question.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: SHIFT, SET

OTHER CRS Plus Publications: SRS-APP/PALLET

SIN

FORMATS:

Source: [Line#] SIN <angle>, <R|D>, <VAR_NAME>
Tokenized: [Line#] /124 <angle>, <R|D>, <VAR_NAME>

DESCRIPTION:

Will determine the Sine of the given angle and store result in radians <R> or degrees <D> as a VAR_NAME.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: COS, TAN, ASIN, ACOS, ATAN

FORMATS:

Source: [Line#] SPEED <Value>
Tokenized: [Line#] /016 <Value>

DESCRIPTION:

The SPEED command instructs the robot to perform motion commands at the specified speed.

The value of the speed corresponds to a percentage of full speed. The range permitted is from 1% to 150% of full speed. 100% speed is the maximum speed at which the robot will move in a coordinated joint-interpolated motion from any one programmable point to any other programmable point. A more detailed discussion of this term can be found in the technical manual.

The speed value can be entered as an explicit value, or as a variable.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: All motion commands

OTHER CRS Plus Publications: SRS-M1A/TECH

SQRT

FORMATS:

Source: [Line#] SQRT <value>, <VAR_NAME>
Tokenized: [Line#] /130 <value>,(VAR_NAME)

DESCRIPTION:

Calculates square root of value and stores result in VAR_NAME.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: !, SIN, COS, TAN, ASIN, ACOS, ATAN

FORMATS:

Source: [Line#] STATUS [1]
 Tokenized: [Line#] /025 [1]

DESCRIPTION:

This command outputs a status display of many important robot operation parameters to the current default device.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Display status, Enter:

>>STATUS<cr>

response:

```

LAST ERROR CODE: 040-ARM POWER
ROBOT SPEED (% of full) : +40
CURRENT PROGRAM EXECUTING: TEST      LINE NUMBER: 00070
PROGRAM LOOPS EXECUTED: 65535      PROGRAM LOOPS REMAINING: 00001
CURRENT PROGRAM EDITING : TEST
JOINT LIMIT CHECK IS ON
ROBOT IS HOMED
TOOL TRANSFORM
NAME      X      Y      Z      Yaw      Pitch      Roll
TOOL_1    +000.000 +000.000 +000.000 +000.000 +000.000 +000.000

```

GRIPPER TYPE : AIR

```

AXIS#  1 2 3 4 5 6 7 8
LOCK   N N N N N
DONE   Y Y Y Y Y
LIMP   Y Y Y Y Y
HOMED  Y Y Y Y Y
CAL    Y Y Y Y Y

```

Configuration Reach Forward/Elbow Up/Invert Off
 Manual Mode ON Mode CYLINDRICAL

CROSS-REFERENCES:

RAPL COMMANDS: none

OTHER CRS Plus Publications: SRS-M1A/TECH

STOP

FORMATS:

Source: [Line#] STOP [message string]
Tokenized: [Line#] /078 [message string]

DESCRIPTION:

This command must be located at the end of a program, unless an ABORT command is used. If it is not present, then an error will occur. The message string is useful for informing the operator when a task is complete.

APPLICABLE MODES:

{P}

EXAMPLE:

1. The result of this portion of code is to terminate the program and print the message 'THAT IS ALL' on the terminal screen.

```
970 MOVE AWAY  
990 STOP THAT IS ALL
```

CROSS-REFERENCES:

RAPL COMMANDS: PAUSE, RUN, RETRY, PROCEED

FORMATS:

Source: [Line#] STRPOS <STR_NUM>, <'TARGET STRING'>, <VAR_NAME>
Tokenized: [Line#] /149 <STR_NUM>, <'TARGET STRING'>, <VAR_NAME>

DESCRIPTION:

Will return the character index of the first character in the TARGET string in the string identified by STR_NUM. The target string must be an assigned text string, bounded by single quotes. If no match exists for the target string, a value of 0 is returned in the variable. Otherwise, a value of 1 to 32 is returned.

APPLICABLE MODES:

{I}, {P}, {M}

EXAMPLE:

1. Assign string 1 with the following data. Pick the index for the target string 'KK' and assign it to variable AA. Print it out.

```
>>! &1 = 'FIND THE KK STRING'  
>>STRPOS 1, 'KK', AA  
>>TYPEI AA  
+10  
>>
```

CROSS-REFERENCES:

RAPL COMMANDS: CUT, PASTE, !, ENCODE, DECODE, IFSTRING are other string related commands.

TAN

FORMATS:

Source: [Line#] TAN <angle>, <R;D>, <VAR_NAME>
Tokenized: [Line#] /126 <angle>, <R;D>, <VAR_NAME>

DESCRIPTION:

Will determine the Tangent of the given angle (in radians <R> or degrees <D>) and store result as a VAR_NAME.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: SIN, COS, ATAN, ASIN, ACOS

FORMATS:

Source: [Line#] TCOMP <LOC_NAME>, <VAR_NAME>
Tokenized: [Line#] /066 <LOC_NAME>, <VAR_NAME>

DESCRIPTION:

The 'T' component (tool or roll angle) of "LOC_NAME" is extracted and stored in "VAR_NAME". The angle is stored in degrees.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: COMP, XCOMP, YCOMP, ZCOMP, OCOMP, ACOMP

FORMATS:

Source: [Line#] TCOMP <LOC_NAME>, <VAR_NAME>
Tokenized: [Line#] /066 <LOC_NAME>, <VAR_NAME>

DESCRIPTION:

The 'T' component (tool or roll angle) of "LOC_NAME" is extracted and stored in "VAR_NAME". The angle is stored in degrees.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: COMP, XCOMP, YCOMP, ZCOMP, OCOMP, ACOMP

TEACH

FORMATS:

Source: [Line#] TEACH <teach template>[,count]
Tokenized: [Line#] /044 <teach template>[,count]

DESCRIPTION:

This command enables the function of the teach pushbutton on the pendant for recording robot positions. Location names will consist of the template and count merged into an 8-character string. The count is incremented by one each time the teach button is pressed. Entering a [count] in the command string permits entry of a new starting count. If not included, the count will continue from the previous count used. The maximum count value is 255. After 255, the counter will reset to 0.

The teach mode remains active until a NOTEACH or DISABLE TEACH command is issued. ENABLE TEACH will re-establish teach button function but not permit changing the template or count values.

WARNINGS:

1. Pressing the TEACH button in the TEACH mode causes a message string to be displayed. Since the teach button has a higher priority than all other normal screen update functions, this message will interrupt dynamic displays such as the one generated by the W1 command.
- 2) The WITH command uses the same internal register for storage of the template as the TEACH command does. Thus the use of either command will overwrite the contents placed in that register by the other command.

APPLICABLE MODES:

{I}, {M}, {P}

...Continued

EXAMPLES:

1. Teach three points, Enter:

```
>>TEACH PICK,10<cr>
```

Press teach button at the desired three locations. Response:

```
>>  
POINT SAVED :  
PICK_010  
>>  
POINT SAVED :  
PICK_011  
>>  
POINT SAVED :  
PICK_012
```

2. Then enter a new template:

```
>>TEACH NEW<cr>
```

Press teach button at the desired location. Response:

```
>>  
POINT SAVED :  
NEW_013  
>>  
POINT SAVED :  
NEW_014  
>>
```

CROSS-REFERENCES:

RAPL COMMANDS: WITH

OTHER CRS Plus Publications: SRS-APP\PATH

TIME

FORMATS:

Source: [Line#] TIME <VAR_NAME>
Tokenized: [Line#] /082 <VAR_NAME>

DESCRIPTION:

Read the system timer clock register. The value returned will be stored in real format in the prescribed variable register. The system timer clock ticks roughly every 41 milliseconds.

The counter is reset only by a TEACH START.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. To determine the time taken for a certain move:

```
...
440 FINISH
450 TIME START
460 MOVE PLACE
470 FINISH
480 TIME FINISH
490 ! TIME = FINISH - START
495 ; To get the value in seconds, multiply by 0.041 seconds/tick:
500 ! TIME = TIME * .041
510 TYPE 'It took '
520 TYPEV TIME
530 TYPE ' seconds for the move.'/
540 ...
```

For realistic numbers, the compiled version of the code would be better as the time to interpret the code would be much less.

CROSS-REFERENCES:

RAPL COMMANDS: DELAY

OTHER CRS Plus Publications: SRS-M1A/TECH

OPEN (/040) - Open the gripper. Force argument used only for optional servo gripper.

OUTPUT (/089) - Set the output lines to the specified states.

PASSWORD (/141) - Enters supervisory mode.

PASTE (/146) - Paste string 1 into string 2, starting at the specified index.

PAUSE (/076) - Pause the program flow and display the message on the output device. The program can then be continued by entering the **PROCEED** command.

POINT (/050) - Defines a location.

PRINT (/055) - Send the text string to the printer. Use a set of special characters to enhance the display. See **TYPE**.

PRINTI (/056) - Print variable(s) to the printer in an integer format.

PRINTV (/057) - Print variable(s) to the printer in real format.

PROCEED (/106) - Continue program execution after a **PAUSE** statement or a <Ctrl-A> has been used to halt program flow. In a program, this command will restart a path once interrupted.

REACH (/171) - Cause to robot to access a singularity location either in forward mode (traditional) or by reaching "over the head".

READY (/015) - Moves arm to the "READY" position, and any extra axes to their zero positions. Also resets any **LOCK** conditions.

RENAME (/107) - Rename an existing program in the robot memory.

RETRY (/108) - Retry any statement which caused an error, halting program flow. That statement will be attempted again and the program will continue normally.

RETURN (/077) - Return to calling program.

RUN (/109) - Execute a program in memory. Execute it for a given number of cycles. Execute it only once if no loop counter is present.

SERIAL (/111) - Display the current serial interface parameters.

SET (/051) - Assign the value of a location to that of an existing one.

SHIFT (/052) - Shift the cartesian displacements of a cartesian location by the specified vector.

- SHIFTA** (/053) - Shift the values of all components of the cartesian location by the specified amounts.
- SIN** (/124) - Determine the sine of the specified angle. The angle can be given in either degrees or radians. Return the result in the given variable location.
- SPEED** (/016) - Set the speed of future robot moves.
- SQRT** (/130) - Determine the square root of the specified value. Store the value of this operation as the specified variable name.
- STATUS** (/025) - Display the current robot status on the default output device.
- STOP** (/078) - Terminate program flow. Print the optional string on the output device.
- STRPOS** (/149) - Match the occurrence of the target string in the specified string number, and return the character index in the variable. Returns a value of zero if no match was made.
- TAN** (/126) - Determine the tangent of the specified angle. The angle can be given in either degrees or radians. Return the result in the given variable location.
- TCOMP** (/066) - Store the tool roll component of a cartesian location as the specified variable name. The angle is stored in degrees.
- TEACH** (/044) - Turn on the TEACH mode of operation. Specify a teach name template which will be used to identify all future teach points. Each time the teach pushbutton is pressed, a new location will be stored in memory. Locations can be stored according to the format described by the name; that is either a precision point, or cartesian location.
- TIME** (/082) - Extract the system timer clock value. Store the value in real format in the given variable.
- TOOL** (/054) - Set the tool transform to a value described by a stored 'location'.
- TRACK** (/180) - Set up robot axis 6 as a Track axis, either in line with the robot X or Y axis. Future cartesian location calls will include the extra axis coordinate.
- TRACE** (/079) - Turn on the trace mode of operation. Each new program line which is executed will be displayed on the output device in terms of program name and line number.
- TRIGGER** (/175) - Set up a table of outputs to be turned on and/or off during path execution.

TYPE (/058) - Send a text string output to the default output device. Use special characters to enhance the display.

TYPEI (/059) - Print the list of variables to the output device in an integer format.

TYPEV (/060) - Print the list of variables to the output device in a real format.

UNLOCK (/017) - Disables LOCK command.

W0 (/027) - Display the current commanded robot position.

W1 (/029) - Display the actual robot position continuously.

W2 (/031) - Display the actual current robot position.

W3 (/033) - Displays the robot commanded position continuously on the output device.

W4 (/035) - Display the end coordinates of the current path.

W5 (/037) - Continuously displays the robot velocity command.

WAIT (/094) - Test the condition of the input line and wait here until the condition is met.

WE1 (/028) - Display the actual position of the extra axes continuously.

WE3 (/032) - Displays the commanded position of the extra axes on the output device continuously.

WE5 (/036) - Displays the extra axes velocity commands continuously.

WGRIP (/041) - Stores the servo gripper opening distance (in inches) in the specified variable.

WITH (/153) - Permits numerical access to all locations stored with the same template.

XCAL (/157) - calibrate an extra axis.

XCOMP (/061) - Stores the X component of a cartesian location (in inches) as the specified variable name.

XHOME (/158) - HOME an extra axis.

XREADY (/014) - Move an extra axis to its zero position.

XZERO (/159) - Zero the position registers of the specified axis.

YCOMP (/062) - Store the Y component of a cartesian location (in inches) as the specified variable name.

ZCOMP (/063) - Store the Z component of a cartesian location (in inches) as the specified variable name.

APPENDIX B - RAPL ERROR LIST

All robot operations are monitored for error conditions. Detectable errors are listed in this appendix. When an error condition exists, a message will be displayed at the default device, and the ready-light on the teach pendant will turn off.

To clear the error condition, any correct command may be entered. The command will clear the error detection flag, and will then turn the ready-light back on. A STATUS command will list the previous error condition.

ERROR MESSAGE	DESCRIPTION
000	<p>AXIS#1 OUT AXIS #1 IS OUT OF RANGE: This error means that there has been a Loss Of FeedBack error for joint 1. This error could be due to a collision with a stationary object, a blown fuse in the motor circuit, or a true loss of feedback signal from the encoder.</p>
001	<p>AXIS#2 OUT AXIS #2 IS OUT OF RANGE: Same as error 000, but for axis 2.</p>
002	<p>AXIS#3 OUT AXIS #3 IS OUT OF RANGE: Same as error 000, but for axis 3.</p>
003	<p>AXIS#4 OUT AXIS #4 IS OUT OF RANGE: Same as error 000, but for axis 4.</p>
004	<p>AXIS#5 OUT AXIS #5 IS OUT OF RANGE: Same as error 000, but for axis 5.</p>
005	<p>AXIS#6 OUT AXIS #6 IS OUT OF RANGE: Same as error 000, but for axis 6.</p>
006	<p>AXIS#7 OUT AXIS #7 IS OUT OF RANGE: Same as error 000, but for axis 7.</p>
007	<p>AXIS#8 OUT AXIS #8 IS OUT OF RANGE: Same as error 000, but for axis 8.</p>
008	<p>ILLEG OPER ILLEGAL OPERAND: An operand in the command line is not acceptable.</p>
009	<p>EOL NOT FO END OF LINE NOT FOUND: The end of line descriptor (a <cr>) was not found, causing an incorrect read of the data. Command lines are limited to 128 characters in length including the EOL character.</p>

ERROR MESSAGE	DESCRIPTION
010 SYMB UNDEF	SYMBOL UNDEFINED: A necessary variable was not present in the symbol table. To correct, manually, or through the program logic, create a symbol in the table.
011 LOCN UNDEF	LOCATION UNDEFINED: The robot location referenced was not previously defined. Check the location table for the location in question, and create if necessary.
012 COMM ERR??	COMMAND ERROR: The command entered was not found in the command list. A spelling mistake, or the lack of a space between the line number and the command descriptor could have caused this problem. Either re-type the line, or inspect the program line for the mistake.
013 ARG MISSIN	ARGUMENT MISSING: An argument was expected, but not read. Check the command syntax.
014 BAD SYNTAX	BAD SYNTAX: A general error which occurs when the command line could not be decoded. Check for a spelling error in the command name.
015 PP NOT LEG	PRECISION POINT NOT LEGAL: A precision point was specified as a location in the command line, but was not permitted in this situation.
016 I/O ADDR??	I/O ADDRESS ERROR: The I/O number which was entered in the command line was beyond the permissible range. Standard digital I/O is from 1 to 16, while optionally, 1 to 40 can be used.
017 PROG NOT F	PROGRAM NOT FOUND: The program called was not found in the program directory. Create the program, or check the spelling of the missing program call command.
018 ILLEG ARGU	ILLEGAL ARGUMENT: An argument was entered, but not permitted.
019 LINE NOT F	LINE NOT FOUND: The program line number called in a branching statement is not present in the program.
020 ARG TOO BI	ARGUMENT TOO BIG: The value of a specified argument is too big for the command.
021 NO ROOM LE	NO ROOM LEFT: The program buffer is full, and cannot accept any further information. Delete unwanted programs, or re-allocate memory with the ALLOCATE command.

ERROR MESSAGE	DESCRIPTION
022 TABLE FULL	TABLE FULL: The robot controller maintains program names, location and variable references in tables. This error means that there is no room left to store a new item. Delete all unwanted information, or re-allocate memory.
023 BAD LINE N	BAD LINE NUMBER: The line number called could not be found, and that a value of zero was returned after its search. The value of zero is an illegal line number.
024 ILL A/D CH	ILLEGAL A/D CHANNEL: An illegal analog to digital conversion channel was specified. Channel numbers 1 to 25 are valid. Channel 9 is the speed selector knob on the teach pendant.
025 LINE TOO L	LINE TOO LONG: The command line exceeded the maximum number of characters permitted, 128 characters.
026 EOF NOT FO	END OF FILE NOT FOUND: An end-of-file character was not found. This implies that a program buffer error has occurred. To remedy clear all memory and re-load all information. If the problem persists, then a fault exists in user memory, or a problem with storing the information on an external device caused invalid storage of information. When this information was re-loaded, the EOF character was not found, thus causing the error.
027 PROG EXIST	PROGRAM EXISTS: An attempt to create a duplicate file name, or an attempt to load into an existing file name was discovered. Either delete the existing file, or define a new program name for the new function.
028 LINEEXIST	LINE EXISTS: An attempt to insert an existing line number was found. Either define a new line number, or delete the old one first.
029 XFRM ILLEG	TRANSFORM ILLEGAL A move to an impossible robot coordinate was attempted and the coordinate transformation for the location generated a mathematically impossible result.
030 RESERVED	

ERROR MESSAGE	DESCRIPTION
031	RESERVED
032	<p>HOME FAIL1 JOINT #1 HOME FAILURE: The home sequence failed during the motion of axis 1. The axis was commanded a motion of just over 1000 pulses, or one motor revolution. During this motion, a zero-crossing marker pulse was expected but not seen. This could be due to an encoder failure, a wiring problem or just a lack of power to motor 1. Check for servo action and retry the HOME command. If it fails again, check encoder function and connection.</p>
033	<p>HOME FAIL2 JOINT #2 HOME FAILURE: Follow the same procedure as error 032.</p>
034	<p>HOME FAIL3 JOINT #3 HOME FAILURE: Follow the same procedure as error 032.</p>
035	<p>HOME FAIL4 JOINT #4 HOME FAILURE: Follow the same procedure as error 032.</p>
036	<p>HOME FAIL5 JOINT #5 HOME FAILURE: Follow the same procedure as error 032.</p>
037	<p>HOME FAIL6 JOINT #6 HOME FAILURE: Follow the same procedure as error 032.</p>
038	<p>HOME FAIL7 JOINT #7 HOME FAILURE: Follow the same procedure as error 032.</p>
039	<p>HOME FAIL8 JOINT #8 HOME FAILURE: Follow the same procedure as error 032.</p>
040	<p>ARM POWER ARM POWER: The arm power switch was turned off, which means that no power is going to the servo motors. In this condition, the robot cannot move. The robot has been placed in a LIMP state.</p>
041	<p>NOT HOMED NOT HOMED: The robot cannot perform the requested motion because it has not been homed after power up.</p>
042	<p>JNT #1 OUT JOINT #1 OUT OF RANGE: The next motion will send joint 1 out of range. Inspect the destination point for validity.</p>
043	<p>JNT #2 OUT JOINT #2 OUT OF RANGE: Follow the same procedure as error 042.</p>

ERROR MESSAGE	DESCRIPTION
044 JNT #3 OUT	JOINT #3 OUT OF RANGE: Follow the same procedure as error 042.
045 JNT #4 OUT	JOINT #4 OUT OF RANGE: Follow the same procedure as error 042.
046 JNT #5 OUT	JOINT #5 OUT OF RANGE: Follow the same procedure as error 042.
047 CHECKSUM F	CHECKSUM FAILURE: The desired item of memory (program, location or variable) has been changed in some unpredictable way. The use of this memory item could cause unexpected results. Check the item and change or edit it to the correct value.
048 RESERVED	
049 MEM NOT AL	MEMORY NOT ALLOCATED: RAPL checked user memory and believes it has not been allocated. This could be an indication of battery failure. Use the ALLOC command to partition and clear the robot memory before use.
050 AMBIG CMD	AMBIGUOUS COMMAND: The user specified too few characters in a command specifier so that it is not yet a unique command choice. Re-enter the command. This error will occur when pressing a <cr> immediately after the prompt.
051 #0 TXD TIM	SERIAL CHANNEL #0 TIMEOUT: A timeout in serial device 0 indicating a hardware communication failure. When you see this message, the condition should already be cleared, but check the cable and its connections to be safe. If this fails, then turn system off and on.
052 #0 CTS TIM	SERIAL CHANNEL #0 CTS TIMEOUT: A timeout in the serial device 0 handshake control signal. This will appear only if the operator has selected a CTS/RTS handshake format. A timeout error would seem to indicate that the external device has failed, or a connection has come loose.
053 #1 TXD TIM	SERIAL CHANNEL #1 TIMEOUT: Follow the same procedure as error 051 but for device 1.
052 #1 CTS TIM	SERIAL CHANNEL #1 CTS TIMEOUT: Follow the same procedure as error 052 but for device 1.
053 RESERVED	

ERROR MESSAGE	DESCRIPTION
054	RESERVED
055	RESERVED
056	CAL CKSUM CALIBRATE CHECKSUM ERROR: The robot arm calibration values stored in memory, which are used in the homing sequence, have been corrupted. Re-Calibrate the robot or re-load the calibration values from disk using the ROBCOMM utility LOADCAL.
057	RESERVED
058	NO ACCESS NO ACCESS: User requires supervisory access to this command. Enter the correct PASSWORD and retry the command.
059	RSERVED IO RESERVED I/O ERROR: Attempt to use serial device #1 when it has been reserved for ACI use.
060	ACI ERROR! REMOTE COMMUNICATION ERROR: Contact your distributor.
061	RESERVED
062	NO RET LEV NO RETURN LEVEL Either a RETURN instruction was used when the program was not called with a GOSUB (ie. trying to RUN a subroutine), or a attempted GOSUB call would result in a subroutine nesting level greater than 10.
063	BAD AXIS N BAD AXIS NUMBER An extra axis command was issued to a nonexistent axis. Use the @NOA command to set the number of axes.
064	SLOT USED AXIS SLOT ALREADY USED Either the programmer was attempting to set up the controller for 8 axes while the servo or magnetic gripper option was installed, or vice versa. The programmer must deactivate the option which uses slot 8 first before assigning the new function.
065	STR ERROR STRAIGHT LINE ERROR A straight line path was halted since it hit a joint limit. It may not be possible to move the arm out of this position using a straight line, so use a JOINT move.
066	NOT INSTAL EXPANSION MEMORY NOT INSTALLED The continuous path command CTPATH was attempted, but the extra memory option was not installed.

APPENDIX C - TERMINAL CONTROL CODES

INTRODUCTION

RAPL contains terminal control codes which can affect robot operation.

To enter a control code, the "control" (usually labeled "Ctrl") key must be pressed while the corresponding UPPERCASE alphabetic key is pressed.

CONTROL CHARACTERS

- <Ctrl-A> The robot will stop after the current block is finished. If a motion is in progress, the next program line is processed as soon as the motion profile has been determined. Stopping a program and motion simultaneously will likely leave the arm out of sync with program execution. <Ctrl-A> is useful since it will terminate program execution but permit the robot to finish motion.
- <Ctrl-C> The robot current operation is terminated immediately. <Ctrl-C> will stop motion, terminate program execution or command line entry, and reset communication parameters. This command is the equivalent of pressing the <ABORT> pushbutton on the pendant.
- <Ctrl-E> Resume echo mode of terminal operation. The serial channel that is affected is the one that received this code.
- <Ctrl-H> Turn on the HELP mode. This will activate the syntax building feature of the operator interface. [NOT effective when using ROBCOMM program.]
- <Ctrl-I> Turn off the HELP mode. This will deactivate the syntax building feature of the control.
- <Ctrl-Q> Resume terminal output. Essentially, this is an XON code that the user can manually issue to resume output to the terminal device.
- <Ctrl-R> The echo mode of terminal operation will be turned off. The serial channel that is affected is the one that received this code.
- <Ctrl-S> Stop terminal output. This code is an XOFF command to the controller that stops further output to the terminal.
- <Ctrl-X> This command is the same as the <Ctrl-C> command. It is intended for use where users utilize a personal computer instead of a terminal to communicate with the robot controller. In this case, often the personal computer responds internally to a <Ctrl-C> character issued from the keyboard.

D-1 INTRODUCTION

Before an auto-start operation is executed the system must have a program called `AUTO_ST`. This special program is executed automatically when the system MAIN POWER is turned on with the AUTO START switch depressed. If this type of start up is attempted and a program with this name is not found, an error will result.

D-2 EXAMPLE 1

This example is an application for the AUTO START procedure using the manual mode to HOME the robot during the start-up.

In this example, the auto start procedure takes care of homing the robot by prompting the user to enter manual mode. The IFSTART, ONSTART and ONPOWER commands are used to synchronize the homing procedure with the user. After homing, program flow will branch to the application program.

Any output to a terminal device will be ignored if no terminal is connected when the system is powered up. The FLASH command can be used to signal the operator in this case. In this case, the teach pendant must be plugged in to the front panel.

AUTO_ST PROCEDURE - example 1

STEP DESCRIPTION

1. Start with the Controller in the OFF condition.
2. Press and HOLD Autostart button.
3. Turn on Controller.
4. Release Autostart button after about three (3) seconds.
5. Watch for Teach Pendant Indicator flashing (Slow).
6. Turn Arm power on.
7. Press START switch again.
8. Watch for Teach Pendant Indicator flashing (Fast).
9. Move robot to home boundaries with teach pendant.
10. Press Autostart button.
11. Homing sequence executes.
12. Application program, `MAIN_PRO`, now executes.

EXAMPLE 1 AUTO_ST PROGRAM LISTING

```
10 TYPE ``/  
20 TYPE `Release START Switch.`/  
30 TYPE ``/  
50 FLASH 5  
60 TYPE `Turn ARM POWER on.`/  
70 TYPE ``/  
80 ONPOWER  
90 FLASH 3  
100 MANUAL  
110 TYPE `Now in MANUAL mode. Move to within HOME`/  
120 TYPE `bounds and press AUTO-START switch.`/  
130 TYPE ``/  
140 ONSTART  
150 NOFLASH  
160 HOME  
170 TYPE `Program will run when AUTO-START switch pressed.`/  
180 FLASH 1  
190 ONSTART  
200 NOFLASH  
210 GOSUB MAIN_PRO  
220 STOP Finished for the day!
```

D-3 EXAMPLE 2

This example assumes the use of the Homing Bracket to position the robot for automatic homing. It also assumes that no terminal will be used for output. In this case, the only operator input will be to turn on the ARM POWER. (For more information on the homing bracket, refer to CHAPTER 9 of the Technical Manual.)

AUTO_ST PROCEDURE - EXAMPLE 2

STEP DESCRIPTION

1. Start with the Controller in the OFF condition.
2. Press and hold AUTO START switch.
3. Turn on Controller.
4. Release AUTO START button after three (3) seconds.
5. Watch for Teach Pendant Indicator flashing (Slow).
6. Turn Arm power on.
7. The robot will home itself and run the application procedure.
8. After completion of the application, the robot will return to the nest and turn off the arm power.

EXAMPLE 2 AUTO_ST PROGRAM LISTING

```
50 FLASH 5
80 ONPOWER
90 NOFLASH
100 ; Exit the homing bracket:
110 SPEED 20
120 JOINT 3,30
130 HOME
140 MOVE SAFE
145 ; Run the application program:
150 GOSUB MAIN_PRO
155 ; When finished, move back to the bracket:
160 MOVE SAFE
170 MOVE ABOVE
180 SPEED 20
190 MOVE BRACKET
200 FINISH
205 ; Turn off the arm power:
210 ARM OFF
220 STOP Finished for the day!
```

Another possibility for an AUTO_ST program is a "Pure executive" program. In such a program, all activities are contained in subroutines. For instance, an auto start program could look like this:

```
100 GOSUB EXIT
200 GOSUB INIT
300 GOSUB MAIN_PRO
400 GOSUB GOODNITE
500 STOP
```

In this case, EXIT waits for arm power, exits the bracket and HOMEs the robot. INIT initializes several system parameters (gripper status, I/O status, robot starting speed, tool transform, WITH command etc.), MAIN_PRO runs the application in a loop form, and GOODNITE puts the robot back into the bracket and turns off arm power.

This program format could be used for any industrial robot application.

FORMATS:

Source: [Line#] TOOL <transform>
 Tokenized: [Line#] /054 <transform>

DESCRIPTION:

To program for different tools of varying geometries, the TOOL transform is provided.

Using the tool transform, the programmer can specify a "tool centre point" of an end effector at some distance away from the centre of the tool mounting flange. The tool transform is generated with any of the location assignment commands and is stored as a location in robot memory.

For quick initiation of the TOOL transform, it is handy to create a location called NULL, which has coordinates: 0,0,0,0,0,0. Then to clear tool transform, enter TOOL NULL.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. To enter a TOOL transform for a gripper with a tool centre point 2 inches from the mounting flange. In Interactive mode, using the POINT command, GRIP_1 is set to 2,0,0,0,0,0:

```
>>POINT GRIP_1 2,0,0,0,0,0<cr>
```

Then under program control:

```
....
```

```
100 TOOL GRIP_1
```

CROSS-REFERENCES:

RAPL COMMANDS: POINT, SHIFT, OFFSET, APPRO, DEPART

OTHER CRS Plus Publications: SRS-M1A/TECH

TRACK

FORMATS:

Source: [Line#] TRACK <X;Y;RESET>
Tokenized: [Line#] /180 <X;Y;RESET>

DESCRIPTION:

This command sets up a TRACK condition. This means that every cartesian location includes the position of axis 6 - the track position.

The coordinate of the track is stored as a real number in 8 bytes immediately above the standard six coordinates of each cartesian entry in the location table. The real value is the "joint" engineering unit value of the axis when the HERE command is issued. Use of @XRATIO, @XLIMITS, and @XPULSES are mandatory before issuing this command. Before storing values or commanding motion after the TRACK command is issued, an XHOME command must have been issued.

The coordinate stored for the extra axis is entered only as an input to the motion command. Shifting of this coordinate is possible using the SHIFTA command. Entry of a point "off-line" using the POINT command will permit entry of the track axis value. All motion commands referring to cartesian locations after issuing the TRACK command will coordinate with the TRACK axis.

For long moves involving the gantry axes it may be an advantage to use the SLEW mode. This is entered with the ENABLE CONSTVEL command.

The TRACK condition is turned off using "TRACK RESET" command.

APPLICABLE MODES:

{I}, {M}, {P}

WARNINGS:

1. Accessing any locations taught without the TRACK option may cause unexpected results.

CROSS-REFERENCES:

RAPL COMMANDS: GANTRY, @XPULSES, @XRATIO, @XLIMITS, POINT, SHIFTA

OTHER CRS Plus Publications: SRS-APP/EXTRA

TRACK

FORMATS:

Source: [Line#] TRACK <X;Y;RESET>
Tokenized: [Line#] /180 <X;Y;RESET>

DESCRIPTION:

This command sets up a TRACK condition. This means that every cartesian location includes the position of axis 6 - the track position.

The coordinate of the track is stored as a real number in 8 bytes immediately above the standard six coordinates of each cartesian entry in the location table. The real value is the "joint" engineering unit value of the axis when the HERE command is issued. Use of @XRATIO, @XLIMITS, and @XPULSES are mandatory before issuing this command. Before storing values or commanding motion after the TRACK command is issued, an XHOME command must have been issued.

The coordinate stored for the extra axis is entered only as an input to the motion command. Shifting of this coordinate is possible using the SHIFTA command. Entry of a point "off-line" using the POINT command will permit entry of the track axis value. All motion commands referring to cartesian locations after issuing the TRACK command will coordinate with the TRACK axis.

For long moves involving the gantry axes it may be an advantage to use the SLEW mode. This is entered with the ENABLE CONSTVEL command.

The TRACK condition is turned off using "TRACK RESET" command.

APPLICABLE MODES:

{I}, {M}, {P}

WARNINGS:

1. Accessing any locations taught without the TRACK option may cause unexpected results.

CROSS-REFERENCES:

RAPL COMMANDS: GANTRY, @XPULSES, @XRATIO, @XLIMITS, POINT, SHIFTA

OTHER CRS Plus Publications: SRS-APP/EXTRA

FORMATS:

Source: **TRACE**
Tokenized: /079

DESCRIPTION:

This command will cause the program and line number being executed to be displayed at the current default device. This is useful when tracing through programs and subroutines for debugging purposes.

Due to the ability of the controller to process a non-motion commands during a move, the line number on the screen is usually the next line after the one being executed. When two consecutive motion commands occur, the line of the second will appear during the execution of the first. This may lead to some confusion.

APPLICABLE MODES:

{I}, {M}

EXAMPLE:

1. To trace program TEST, Enter:

```
>>TRACE  
>>RUN TEST
```

response:

```
TEST [00030]  
TEST [00040]
```

etc.

CROSS-REFERENCES:

RAPL COMMANDS: NOTRACE, ENABLE, DISABLE, NEXT

TRIGGER

FORMATS:

Source: [Line#] TRIGGER [<TRIG NUM>, <+/-OUTPUT NUM>, <LOCATION>]
Tokenized: [Line#] /175 [<TRIG NUM>, <+/-OUTPUT NUM>, <LOCATION>]

DESCRIPTION:

Triggers are used in conjunction with CPATH and CTPATH commands. Triggers activate outputs when the robot passes through set points in a path, permitting digital output states to be changed without interrupting motion. Using the - character before the output number will specify a low level for that particular output, similar to the OUTPUT command. Due to the internal architecture of RAPL, timing of the output is accurate to within 40 milliseconds of passing the specified location.

The trigger table can contain up to 8 entries. A complete entry in the table consists of location name, output number and the state to which the output will be changed at the specified location. Locations can be precision points or cartesian locations.

On power-up, the trigger table is cleared and the trigger flag is disabled. It is necessary to set up the trigger table in any program which makes use of this function. Triggers will not be active until after the ENABLE TRIGGER command has been executed. The TRIGGER table must be set up before calling the path command intending to use the triggers; the CTPATH or CPATH commands.

Entering a 0 for the trigger number will place the new entry in the first vacant spot in the trigger table. Entering a 0 for the output number will clear that trigger table entry. Entering a '0<cr>' after the command will provide a display of the current trigger settings.

APPLICABLE MODES:

{P}, {I}, {M}

EXAMPLE:

1. To register a trigger table entry:

```
>>TRIGGER 1,3,POINT1
```

This will turn output 3 high when the robot passes POINT1 in the path.

...Continued

2. To register a trigger at the next available spot in the table:

```
>>TRIGGER 0,-3,POINT2<cr>
>>CPATH POINT1,POINT2,POINT3<cr>
```

This will turn output 3 off at POINT2.

3. To display the current trigger table:

```
>>TRIGGER<cr>
Trigger #(1-8), 0 for next entry, or <CR> for trigger list:
***** Trigger List *****
Trig# Output# Location State
001 003 POINT1 HI
002 003 POINT2 LO
003**** No Valid Trigger Loaded *****
004**** No Valid Trigger Loaded *****
005**** No Valid Trigger Loaded *****
006**** No Valid Trigger Loaded *****
007**** No Valid Trigger Loaded *****
008**** No Valid Trigger Loaded *****
```

4. To clear the first entry:

```
>>TRIGGER 1,0,POINT1<cr>
```

5. To enable the trigger the trigger for any subsequent path moves:

```
>>ENABLE TRIGGER<cr>
```

CROSS-REFERENCES:

RAPL COMMANDS: CTPATH, OUTPUT, ENABLE, DISABLE

OTHER CRS Plus Publications: SRS-APP/PATH

TYPE

FORMATS:

Source: [Line#] TYPE <'text'&Num>[argument]
Tokenized: [Line#] /058 <'text'&Num>[argument]

DESCRIPTION:

Types a line of text characters on the terminal (device 0). The following are valid arguments:

- / Include a carriage return/line feed combination after the text
- * Include only a carriage return after the text.
- B Output the BELL character to the display.
- F Output a form feed to the device.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Type TEST on terminal.
>>TYPE 'TEST'
2. Type TEST on terminal with a carriage return and line feed.
>>TYPE 'TEST'/'
3. Type TEST on terminal with a carriage return.
>>TYPE 'TEST'*
4. Type TEST on terminal with a bell.
>>TYPE 'TEST'B
5. Type a string on the terminal
>>TYPE &1

CROSS-REFERENCES:

RAPL COMMANDS: TYPEI, TYPEV, PRINT, PRINTI, PRINTV

FORMATS:

Source: [Line#] TYPEI <VAR_NAME>[,...]
Tokenized: [Line#] /059 <VAR_NAME>[,...]

DESCRIPTION:

TYPEI commands are automatically routed to the serial device #0, as a terminal operation.

Often it is useful to display variables in an integer format, since a variable may only represent an integer value. A counter which must be displayed each loop looks far more informative when shown as a '+1' instead of '+1.0000' as a real variable would be displayed. If the value does not fit within the integer limits (+/- 32766) then the display will be shown in real format.

Each typed field will be separated by a single space. No carriage return or line feed control is included in this command, since it is often useful to display more than one value on a display line.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: TYPE, TYPEV, PRINT, PRINTI, PRINTV

TYPEV

FORMATS:

Source: [Line#] TYPEV <VAR_NAME>[,...]
Tokenized: [Line#] /060 <VAR_NAME>[,...]

DESCRIPTION:

TYPEV commands are automatically routed to the serial device #0, as a terminal operation.

Each typed field will be separated by a single space. No carriage return or line feed control is included in this command, since it is often useful to display more than one value on a display line.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Types variable value RADIUS to video terminal.

```
>>TYPEV RADIUS  
+002.7500
```

2. Types variable value RADIUS and CIRCUM to video terminal.

```
>>TYPEV RADIUS,+002.7500 CIRCUM  
+008.6393  
>>
```

CROSS-REFERENCES:

RAPL COMMANDS: TYPE, TYPEI, PRINT, PRINTI, PRINTV

FORMATS:

Source: [Line#] UNLOCK <joint#>[,...]
 Tokenized: [Line#] /017 <joint#>[,...]

DESCRIPTION:

The specified joints will be controllable, as usual with the regular set of motion commands.

The FINISH command will be able to synchronize any robot command with a motion of a previously LOCKed joint. It is recommended that the LOCK command be used only until both sets of motion have been commanded. Issuing the UNLOCK command will then re-establish full communication with all robot joints. Issuing the UNLOCK command will not disturb any motion in progress.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLES:

1. The following example is an excerpt from a program which commands the robot to load a rotary table (Joint #6). The rotary table then moves to a new position, while the robot proceeds to pick up a new part.

```

100 MOVE PICK,S
105 CLOSE
110 DEPART 2
115 ; Move to the rotary table
120 APPRO TABLE,2
130 MOVE TABLE,S
140 OPEN
150 DEPART 2
155 ; Command the table to rotate at speed 20 by 180 degrees
160 SPEED 20
165 UNLOCK 6
170 JOINT 6,180
175 ; Lock out the table from the next move commands
180 LOCK 6
200 APPRO PICK,2
210 GOTO 100

```

CROSS-REFERENCES:

RAPL COMMANDS: LOCK, JOINT

OTHER CRS Plus Publications: SRS-APP/EXTRA

W0

FORMATS:

Source: [Line#] W0
Tokenized: [Line#] /027

DESCRIPTION:

Displays the current robot position in the motor, joint and world coordinate systems. If more than the standard 5 axes are installed, the extra axes will be displayed under the first, second and third values in the MOTOR and JOINT displays. In the WORLD display, they will be displayed only if a TRACK or GANTRY command has been issued.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. In the case where a GANTRY or TRACK command has been entered:

>>W0

COMMANDED POSITION :

NAME	AX#1/6	AX#2/7	AX#3/8	AX#4	AX#5	
PULSES	+0000000000	-000001B000	+0000000000	+0000000000	+0000000000	
	+0000000000	-0000010000				
NAME	JT#1/6	JT#2/7	JT#3/8	JT#4	JT#5	
JOINTS	+000.0000	+090.0000	+000.0000	+000.0000	+000.0000	
	+000.0000	-002.0000				
NAME	X/TRACKX	Y/TRACKY	Z	Yaw	Pitch	Roll
WORLD	+013.5000	+000.0000	+020.0000	+000.0000	+000.0000	+000.0000
	+000.0000	-002.0000				

CROSS-REFERENCES:

RAPL COMMANDS: W1, W2, W3, W4, W5, WE1, WE3, WE5, LISTL, HERE

FORMATS:

Source: [Line#] W1
 Tokenized: [Line#] /029

DESCRIPTION:

Continually displays the actual robot position in motor coordinates. A <cr> or an <ABORT> is required to terminate this command.

If used in a program, program flow will wait here until a <cr> is entered from the keyboard.

WARNINGS:

1. In use with the ROBCOMM software terminal emulation mode, an error or pressing the TEACH button to register a location during a dynamic display will cause the screen to lock up. Use caution when these displays are underway.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Enter:

>>W1

ACTUAL POSITION (MOTOR PULSES):

+0000000001 -0000000003 +0000000031 +0000000000 +0000000000

CROSS-REFERENCES:

RAPL COMMANDS: W0, W2, W3, W4, W5, WE1, WE3, WE5, LISTL, HERE

W2

FORMATS:

Source: [Line#] W2
Tokenized: [Line#] /031

DESCRIPTION:

Displays the current actual robot position in motor, joint and cartesian coordinates. If more than the standard 5 axes are installed, the extra axes will be displayed under the first, second and third values in the MOTOR and JOINT displays. In the WORLD display, they will be displayed only if a TRACK or GANTRY command has been issued.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Enter:

>>W2

CURRENT ACTUAL POSITION:

NAME	AX#1/6	AX#2/7	AX#3/8	AX#4	AX#5	
PULSES	+0000000000	-000001B000	+0000000000	+0000000000	+0000000000	
	+0000000000	-0000010000				
NAME	JT#1/6	JT#2/7	JT#3/8	JT#4	JT#5	
JOINTS	+000.0000	+090.0000	+000.0000	+000.0000	+000.0000	
	+000.0000	-002.0000				
NAME	X/TRACKX	Y/TRACKY	Z	Yaw	Pitch	Roll
WORLD	+013.5000	+000.0000	+020.0000	+000.0000	+000.0000	+000.0000
	+000.0000	-002.0000				

CROSS-REFERENCES:

RAPL COMMANDS: W0, W1, W3, W4, W5, WE1, WE3, WE5, LISTL, HERE

FORMATS:

Source: [Line#] W3
 Tokenized: [Line#] /033

DESCRIPTION:

Continually displays the commanded robot position in motor coordinates. A <cr> or an <ABORT> is required to terminate this command.

If used in a program, program flow will wait here until a <cr> is entered from the keyboard.

WARNINGS:

1. In use with the ROBCOMM software terminal emulation mode, an error or pressing the TEACH button to register a location during a dynamic display will cause the screen to lock up. Use caution when these displays are underway.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Enter:

>>W3

COMMANDED POSITION (MOTOR PULSES):

+0000000001 -0000000003 +0000000031 +0000000000 +0000000000

CROSS-REFERENCES:

RAPL COMMANDS: W0, W1, W2, W4, W5, WE1, WE3, WE5, LISTL, HERE

W4

FORMATS:

Source: [Line#] W4
Tokenized: [Line#] /035

DESCRIPTION:

Displays the current motion end point in motor, joint and cartesian coordinates. If more than the standard 5 axes are installed, the extra axes will be displayed under the first, second and third values in the MOTOR and JOINT displays. In the WORLD display, they will be displayed only if a TRACK or GANTRY command has been issued.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Enter:

>>W4

NEXT END POINT:

NAME	AX#1/6	AX#2/7	AX#3/8	AX#4	AX#5	
PULSES	+0000000000	-0000018000	+0000000000	+0000000000	+0000000000	
	+0000000000	-0000010000				
NAME	JT#1/6	JT#2/7	JT#3/8	JT#4	JT#5	
JOINTS	+000.0000	+090.0000	+000.0000	+000.0000	+000.0000	
	+000.0000	-002.0000				
NAME	X/TRACKX	Y/TRACKY	Z	Yaw	Pitch	Roll
WORLD	+013.5000	+000.0000	+020.0000	+000.0000	+000.0000	+000.0000
	+000.0000	-002.0000				

CROSS-REFERENCES:

RAPL COMMANDS: W0, W1, W2, W3, W5, WE1, WE3, WE5, LISTL, HERE

FORMATS:

Source: [Line#] W5
Tokenized: [Line#] /037

DESCRIPTION:

Continually displays the robot velocity command. A <cr> or an <ABORT> is required to terminate this command.

If used in a program, program flow will wait here until a <cr> is entered from the keyboard.

WARNINGS:

1. In use with the ROBCOMM software terminal emulation mode, an error or pressing the TEACH button to register a location during a dynamic display will cause the screen to lock up. Use caution when these displays are underway.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Enter:

>>W5

VELOCITY COMMAND (MOTOR PULSES):
+0 +0 +0 +0 +0

CROSS-REFERENCES:

RAPL COMMANDS: W0, W1, W2, W3, W4, WE1, WE3, WE5, LISTL, HERE

WAIT

FORMATS:

Source: [Line#] WAIT <[-]input#>
Tokenized: [Line#] /094 <[-]input#>

DESCRIPTION:

This command will halt program flow until the value of the specified input point matches the required state. The input value can range from 1 to 40 with the extended I/O option, but 1 to 16 is standard.

The sense of the input is denoted by including a sign with the input number. If the input number is negative, then a low level on the input is requested. Likewise, a positive sign, or none at all, implies that a high state on the input is required.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Wait until input #2 is at a low level, then continue.
100 WAIT -2
2. Wait until input #2 is at a high level, then continue.
120 WAIT 2

CROSS-REFERENCES:

RAPL COMMANDS: IFSIG, ONSIG

OTHER CRS Plus Publications: SRS-APP/SAFETY

FORMATS:

Source: [Line#] WE1
Tokenized: [Line#] /028

DESCRIPTION:

Continually displays the actual position of the extra axes in motor coordinates.

A <cr> or an <ABORT> is required to terminate this command.

If used in a program, program flow will wait here until a <cr> is entered from the keyboard.

WARNINGS:

1. In use with the ROBCOMM software terminal emulation mode, an error or pressing the TEACH button to register a location during a dynamic display will cause the screen to lock up. Use caution when these displays are underway.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Enter:

>>WE1

ACTUAL POSITION (motor pulses):
+000002789 +123456789 -987654321

CROSS-REFERENCES:

RAPL COMMANDS: W0, W1, W2, W3, W4, W5, WE3, WE5, LISTL, HERE

WE3

FORMATS:

Source: [Line#] WE3
Tokenized: [Line#] /032

DESCRIPTION:

Continually displays the commanded position of the extra axes in motor coordinates.

A <cr> or an <ABORT> is needed to terminate the display.

If used in a program, program flow will wait here until a <cr> is entered from the keyboard.

WARNINGS:

1. In use with the ROBCOMM software terminal emulation mode, an error or pressing the TEACH button to register a location during a dynamic display will cause the screen to lock up. Use caution when these displays are underway.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Enter:

>>WE3

COMMANDED POSITION (motor pulses):
+000002789 +123456789 -987654321

CROSS-REFERENCES:

RAPL COMMANDS: W0, W1, W2, W3, W4, W5, WE1, WE5, LISTL, HERE

FORMATS:

Source: [Line#] WE5
Tokenized: [Line#] /036

DESCRIPTION:

Continually displays the velocity commands to the extra axis motors.
115 is the maximum command allowed.

A <cr> or an <ABORT> will terminate this command.

If used in a program, program flow will wait here until a <cr> is entered from the keyboard.

WARNINGS:

1. In use with the ROBCOMM software terminal emulation mode, an error or pressing the TEACH button to register a location during a dynamic display will cause the screen to lock up. Use caution when these displays are underway.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. Enter:

>>WE5

VELOCITY COMMAND (MOTOR PULSES):

+5 -12 +65

CROSS-REFERENCES:

RAPL COMMANDS: W0, W1, W2, W3, W4, W5, WE1, WE3, LISTL, HERE

(CAN NOT BE RUN FROM THE EMULATION TERMINAL
MUST BE USED WITHIN A PROGRAM).

WGRIP

EXAMPLE: >> EDIT Blocks

>> INSEKT 520 X=WGRIP()

(X = YOUR VARIABLE FOR
GRIP ENCODER VALUE.

FORMATS:

Source: [Line#] WGRIP <VAR_NAME>
Tokenized: [Line#] /041 <VAR_NAME>

↳ Line# (see Pg. 1-2)

DESCRIPTION:

To read the position of the fingers of the servo gripper. The value of the opening in inches or millimetres will be placed in the variable identified by the argument.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: GRIP, OPEN, CLOSE, @@GTYPE

OTHER CRS Plus Publications: SRS-SGRIP/MAN

FORMATS:

Source: [Line#] WITH <TEMPLATE>
 Tokenized: [Line#] /153 <TEMPLATE>

DESCRIPTION:

This command permits the operator to set up an implied addressing scheme for locations stored in memory using the TEACH command. The template name entered in this command adheres to the same programming rules as does the template in the TEACH command (see section 1-5).

Once the WITH command has specified a template, then the programmer need only to use the ^ character to address the range of locations stored under that template.

Since an index number must be associated with a template before any location can be uniquely defined, the ^ operator permits the use of a number or variable for this purpose.

WARNING:

The WITH command uses the same internal register for storage of a template as does the TEACH command, so that invoking either command will overwrite the contents placed in that register by the other command.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. A program may use the WITH command as follows:

```
100 WITH POINT
120 MOVE ^O
130 ! C = 0
140 MOVE ^C
```

Lines 120 and 140 both create a MOVE command to location POINT000 stored in memory. The advantage of using a variable as an index (as in line 140) is that the same MOVE command can be used to move the robot to any one of the taught points that use the template specified by the WITH command. The value of the variable can be calculated using logic.

CROSS-REFERENCES:

RAPL COMMANDS: TEACH, NOTEACH, CTPATH

OTHER CRS Plus Publications: SRS-APP/PATH

XCAL

FORMATS:

Source: [Line#] XCAL <Axis#>
Tokenized: [Line#] /157 <Axis #>

DESCRIPTION:

The XCAL command permits the programmer to store a HOME position for any extra axis (6, 7 or 8). Subsequent HOME commands will use the absolute position stored by this command. The homing mark is the next encoder zero-cross detection. The motor will be moved in a positive rotation (counter-clockwise) until the mark is reached.

APPLICABLE MODES:

{I}, {P}, {M}

EXAMPLE:

1. Using XZERO command, zero the axis in question.
2. Using the manual mode, or a JOINT or motor command, move the axis to within one (1) revolution of the homing mark.
3. Enter the XCAL command.
4. The axis is now calibrated.

CROSS-REFERENCES:

RAPL COMMANDS: XZERO, XHOME, XREADY

OTHER CRS Plus Publications: SRS-APP/EXTRA

FORMATS:

Source: [Line#] XCOMP <LOC_NAME>, <VAR_NAME>
Tokenized: [Line#] /061 <LOC_NAME>, <VAR_NAME>

DESCRIPTION:

The X coordinate component in "LOC_NAME" is extracted and stored in "VAR_NAME".

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: YCOMP, ZCOMP, ACOMP, OCOMP, TCOMP, COMP

OTHER CRS Plus Publications: SRS-APP/PALLET

XHOME

FORMATS:

Source: [Line#] XHOME <axis#>
Tokenized: [Line#] /158 <axis#>

DESCRIPTION:

Similar to the HOME function, the XHOME command will home a single extra axis. (Axis #6, 7 or 8). When the homing point has been reached, the value of the calibrated position will be loaded into the position registers.

APPLICABLE MODES:

{I}, {P}, {M}

EXAMPLE:

1. Enter manual mode:
>>>MANUAL
2. Move the extra axis to within 1 motor revolution of the homing position.
3. Enter the XHOME command, say, for axis #6:
>>>XHOME 6
4. Respond with a 'Y' to the query.
5. The control will home the specified axis.

CROSS-REFERENCES:

RAPL COMMANDS: XZERO, XCAL, XREADY

OTHER CRS Plus Publications: SRS-APP/EXTRA

FORMATS:

Source: [Line#] XREADY <Axis#>
Tokenized: [Line#] /014 <Axis#>

DESCRIPTION:

This command will move the specified extra axis to its zero position.

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: XZERO, XCAL, XHOME

OTHER CRS Plus Publications: SRS-APP/EXTRA

XZERO

FORMATS:

Source: [Line#] XZERO <Axis#>
Tokenized: [Line#] /159 <Axis#>

DESCRIPTION:

This command will zero out the position registers for the specified axis. This is useful when doing a calibration sequence for an axis, as it allows a known position to be loaded into the position registers.

APPLICABLE MODES:

{I}, {M}, {P}

EXAMPLE:

1. To enter a value of zero (0) into the current position register for axis six (6):

```
>>XZERO 6
```

CROSS-REFERENCES:

RAPL COMMANDS: XZERO, XHOME, XREADY

OTHER CRS Plus Publications: SRS-APP/EXTRA

FORMATS:

Source: [Line#] YCOMP <LOC_NAME>, <VAR_NAME>
Tokenized: [Line#] /062 <LOC_NAME>, <VAR_NAME>

DESCRIPTION:

The Y coordinate component of "LOC_NAME" is extracted and stored in "VAR_NAME".

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: XCOMP, ZCOMP, ACOMP, OCOMP, TCOMP, COMP

OTHER CRS Plus Publications: SRS-APP/PALLET

ZCOMP

FORMATS:

Source: [Line#] ZCOMP <LOC_NAME>, <VAR_NAME>
Tokenized: [Line#] /063 <LOC_NAME>, <VAR_NAME>

DESCRIPTION:

The Z coordinate component of "LOC_NAME" is extracted and stored in "VAR_NAME".

APPLICABLE MODES:

{I}, {M}, {P}

CROSS-REFERENCES:

RAPL COMMANDS: XCOMP, YCOMP, ACOMP, OCOMP, TCOMP, COMP

OTHER CRS Plus Publications: SRS-APP/PALLET

5-1 DESCRIPTION

RAPL provides the programmer with a line editing feature which can be used to create or modify existing programs. The EDIT command is used to access this editor. If a program name is given in the EDIT command, then that program becomes the object of all future EDIT commands and immediate mode line entries until a different program is edited. If no program by that name exists, The editor creates one.

To shorten the development time of a program, RAPL allows the programmer to insert program lines while outside of the standard EDIT mode. While in the {I} or {M} control mode, entering a number before a command identifier will automatically place the following line into the current program being edited. No syntax check is performed on the line before it is inserted. A valid line number must be entered, and the target program for the edit mode must have already been defined.

A program line must always start with a valid line number. Valid line numbers are from 1 to 65536. Entering a line number larger than 65536 will result in an entry equivalent to: $([\text{line\#}] \bmod 65536)$ even though it may be displayed as entered. Entering a line number of 0 will result in an error.

A line is inserted into a program by typing the line number, followed by a command line. When the carriage return is entered, the line is complete, and is entered into the program in the correct numerical sequence. The program memory is updated to reflect the change.

A line cannot be entered if a line with the same line number exists. Thus changing a line requires that it be deleted first.

5-2 EDIT COMMANDS

Once in the edit mode, several commands are available. Each is entered with a single keystroke; RAPL will echo the complete command. Each command letter must be directly followed by the line number where applicable. The commands are as follows:

Copy

It may be necessary to use identical commands in a program. In order to reduce the number of keystrokes required to enter these lines, the Copy command can be used as shown.

Format: C<Line#>,<New Line#>
Example: Copy100,200

Delete

A program line is deleted. The deleted program line can not be recalled.

Format: D<Line#>
Example: Delete100

End

Exits the line editor mode.

Format: E
Example: End

Insert

A new command line is inserted into the program. A line cannot be inserted if its line number already exists.

Format: I<Line#> <Command> <Arguments>
Example: Insert100 MOVE POINT

List

The programmer can display a line of the program while in the edit mode with this command. It can be useful when modifying the program to first view the line which will be affected.

Format: L<Line#>
Example: List100

Move

It may be necessary to move a line from one place to another. When the line is assigned a new number, it is automatically removed from the program and replaced in the new correct numerical order.

Format: M<Old Line#>,<New Line#>
Example: Move100,200

? (/000) - RAPL Command directory

; (/098) - Comment statement.

! (/070) - Assign the value of the simple expression to the variable (/000) specified on the left side of the =.

ABORT (/071) - Terminate existing program.

ACOMP (/065) - Store the A component of a cartesian location as the specified variable name. This is the azimuthal angle. The angle is stored in degrees.

ACOS (/128) - Determine the arc cosine of the specified value. Store the result in either degrees or radians as the specified variable name.

ACTUAL (/152) - Stores the actual position of the arm.

ALIGN (/001) - Align the tool axis with the nearest world major axis (X, Y, or Z).

ALLOC (/018) - Allocate robot memory according to an Automatic format, or a user defined format. In this way, robot memory can be divided between locations, variables and robot programs as the programmer chooses.

ANALOG (/083) - Read the specified analog input channel, and store the unipolar value as an integer number (value between 0 and 255) in the specified variable location.

AOUT (/093) - Send a value to an analog output point (included in the COMBO card option). The digital value is from -5 to +5 corresponding to digital 0 to 255.

APPRO (/002) - Approach the given point (either cartesian or precision point) by the specified amount. The approach path can be defined as straight by using the optional [S] argument.

ARM (/142) - Enables or disables the arm power. Can be used to turn arm power off but not back on.

ASIN (/127) - Determine the arc sine of the specified value. Store the result in either degrees or radians as the specified variable name.

ATAN (/129) - Determine the arc tangent as specified by the value of Y/X. Store the result in either degrees or radians as the specified variable name. The angle returned will be an absolute value of 0 to 359.9999 degrees or the equivalent in radians.

CLOSE (/038) - Close the gripper. Force argument used only for optional servo gripper.

COMP (/181) - Extract a component out of a stored location. All cartesian components supported, as well as precision points.

CONFIG (/110) - Set the configuration of one of the two serial input channels to the required baud rate, parity, etc.

COPY (/099) - Copy one program to another.

COS (/125) - Determine the cosine of the specified angle. The angle can be given in either degrees or radians. Return the result in the given variable.

CPATH (/163) - Execute a continuous path through the points specified in the argument list. Up to 16 locations can be specified; cartesian and precision points cannot be mixed.

CTPATH (/164) - Calculate a continuous path through the selected series of points that are identified by the template name, with the path passing through all those points with TEACH index numbers in the specified range at the specified speed. Path is executed with GOPATH.

CUT (/145) - Cut characters out from a string.

DECODE (/147) - Decode a real value from the string, starting at the character index, and load the value into the variable.

DELAY (/081) - Specify a time delay, in seconds. The resolution of this command is in milliseconds.

DELETE (/100) - Delete program.

DEPART (/004) - Depart from the present location (along the TOOL axis) by a specified amount. The optional [S] argument will specify a straight line motion.

DEVICE (/112) - Select either device 0 or device 1 as the future default input/output device for user interaction.

DIR (/102) - Send a list of all programs, and the corresponding memory requirements to the output device. The optional argument will send the output to the printer port.

DISABLE (/177) - Turn off a software feature.

DLOCN (/048) - Delete location(s) from the robot's memory.

DVAR (/067) - Delete variable(s) from the robot's memory.

EDIT (/101) - Assign the specified program name for all future edit commands and enter edit mode. The program will be created if it does not already exist.

ELBOW (/169) - Specify the position of the elbow (joint #3) for future moves. Allows elimination of singularity locations.

ENABLE (/176) - Turn on a software feature.

ENCODE (/148) - Encode a value into the string, in either real or integer format.

FINISH (/010) - Complete current motion command before continuing.

FLASH (/084) - Flash the READY lamp on the teach pendant at the specified frequency. Valid range is 1 to 255, 1 is fastest.

FREE (/023) - Display the current status of robot memory.

GAIN (/150) - Permits the user to change the position gain of each motor servo system. A motor number of 0 will set all motors.

GANTRY (/180) - Set up the use of robot axes 6 and 7 as gantry axes X and Y. All locations will include gantry coordinates.

GOPATH (/165) - Execute a path calculated with the CTPATH command.

GOSUB (/074) - Send program control to the specified subroutine. If a list of arguments exist, then replace the current set of 8 macro parameters with the contents of the list. These parameters are identified by variable names %0 to %7.

GOTO (/075) - Send program control to the specified line number in the current program.

GRIP (/039) - Commands the opening or closing of the servo gripper to a specified distance.

HALT (/154) - Halt robot motion or set up the HOLD input.

HELP (/095) - Enables syntax building feature.

HERE (/024) - Stores the current robot commanded position as a precision point or cartesian location in the location table.

HOME (/020) - Send the robot to its home position, assuming that the robot arm is already within the correct bounds of the home position. Replace the robot command registers with the calibrated home position when it is reached.

IF (/072) - Evaluate the logical expression according to one of 6 operators. If the expression is true, then send program control to the specified line number.

IFPOWER (/155) - This a special case of IFSIG, here the arm power status is sampled.

IFSIG (/073) - Examine the states of the specified inputs. If all conditions are true, then send program control to the given line number.

IFSTART (/086) - Examine the state of the autostart pushbutton on the front panel. If it is set, then branch to the specified line number.

IFSTRING (/151) - Compare two strings and branch on result.

IGNORE (/087) - Cancel ONSIG command.

INPUT (/068) - Accept input from the current terminal device.

INVERT (/170) - Invert the coordinate system of the robot.

JOG (/005) - Move the robot end effector by the specified cartesian offsets in a straight line fashion.

JOINT (/006) - Move the specified joint by the given number of units.

LIMP (/007) - Limp specified robot joint. No 'joint#' entered will limp all joints. Terminated by NOLIMP command.

LISTL (/049) - List all location values in a tabular format on the output device. The 1 argument will send output to the printer.

LISTP (/103) - List the program on the output device. Send it to the printer if the 1 argument is specified.

LISTV (/069) - List the variables on the output device. Send it to the printer if the 1 argument is specified.

LOCK (/009) - Prevents selected joints from motion.

MA (/132) - Move all 5 joints to an absolute radian position.

MAGGRIP (/042) - Adjust the magnetic strength.

MANUAL (/045) - Activate the manual control mode, where the teach pendant can be used to control robot motion either by axis or in cylindrical form.

MI (/131) - Move all 5 joints by an incremental radian value.

MOTOR (/011) - Command an individual motor to move by a specified number of pulses.

MOVE (/012) - Move the robot to the specified cartesian or precision point location using the speed as given in the **SPEED** command. The optional **S** argument will command a straight line move to the point.

NEW (/021) - Erase all robot user memory. All programs, variables and locations will be erased.

NEXT (/105) - Execute the next program line. Program name and line number may optionally be specified.

NOFLASH (/088) - Cancel **FLASH** command. **READY** lamp on the teach pendant indicates robot alarm status.

NOHELP (/096) - Disables syntax builder.

NOLIMP (/008) - Re-engages positional servo for specified joint or all joints if no joint# specified.

NOMANUAL (/046) - Disables teach pendant.

NOTEACH (/043) - Turn off the **TEACH** mode.

NOTRACE (/080) - Turn off the **TRACE** mode.

OCOMP (/064) - Store the **YAW** component of a cartesian location as the specified variable name. The angle is stored in degrees.

OFFSET (/047) - Set the robot base offset to the specified displacements and yaw angle. All future locations will be made relative to this new base offset.

ONERR (/143) - Program flow will enter subroutine "**PRG_NAME**" instead of going to the prompt in the case of a **RAPL** error.

ONPOWER (/156) - This a special case of **WAIT**, here the arm power status is sampled.

ONSIG (/090) - Set the specified input line as a special input which will be monitored at a regular interval. When the input changes to the desired state, program flow is re-directed to the specified program. Returning from this program will place the **RAPL** program pointer at the first program line after the one in which the signal was detected.

ONSTART (/092) - Sample the state of the autostart push-button. When it is switched high, program control will continue to the next line. Wait at this point until the condition is met.

B-1 INTRODUCTION

The ACI permits external computer systems to communicate with one or more robot controllers on a single RS232 link.

In its basic configuration, this protocol is used to transfer raw data either to or from the master device. The protocol allows any chunk of 8086 memory to be the object of the communication in a segmented addressing technique.

The protocol permits error checking and automatic transmission retries in order to establish a communication link.

B-2 FUNCTIONAL DESCRIPTION

The ACI is a master/slave protocol. All robot controllers are configured as slaves in the network. Any external computer would then have to be a master. Only a master can establish a communication. All the communication initiative must be taken by the master unit.

B-3 MASTER PROTOCOL

The master device must establish a communication link by specifying a slave device number as a target. It must then confirm to the target that it is indeed requesting a communication. After the link has been established, the master must then provide the slave with the information describing the data transfer that is about to take place. After all of the particulars concerning the communication has been transferred, the actual data is then sent. The data is transferred in packages of 128 bytes. Each 'block' of data contains its own start/stop characters and a checksum test byte to validate the data after transfer. Finally, it is up to the master to close the communication link with an EOT character.

A communication consists of four separate blocks. They are described in detail in the following sections.

B-4 ENQUIRY SEQUENCE

The master issues a simple three byte code to the serial line. This code is;

'R' , slave ID + 20h , ENQ

The master must then follow this sequence with two character times of no transmission. This will ensure that the slave has indeed read a valid enquiry sequence, and not some random chunk of another communication. When the slave has interpreted a correct enquiry, it will issue an appropriate response;

'R' , slave ID + 20h , ACK

When the master reads this response it has established the communication link to the target device.

The slave id number is any number between 0 and 7F hexadecimal. Using the ACI monitor command @@RN, the programmer can configure each robot controller with an appropriate slave id number.

Adding a value of 20 hex to the slave number provides security that the second byte of this string does not resemble a control code in any way. This way, the slave device will be able to distinguish an enquiry sequence and can establish communications quickly.

The master should attempt to contact the slave only a limited number of times before abandoning the communication. A failure to establish the communications with the slave after three attempts normally indicates a failure in the link. Either the baud rates are not set correctly at either end, or a physical problem exists with the interface wiring.

A delay of 2 character times should be included in the transmission of the enquiry sequence. This delay is placed between the second and third characters of the enquiry sequence. This time delay ensures that a three byte string from a data block cannot be misconstrued as an enquiry sequence. The master control must ensure that this time delay exists, or proper ACI operation cannot be guaranteed.

B-5 HEADER SEQUENCE

The header block consists of a description of the data transfer that will take place. The header block is broken down into the following byte description;

1	SOH	Start of header character
2	SLAVE ID + 20h	Slave identification number
3	MASTER ID + 20h	master identification number
4	READ/WRITE	data read or write selection
5	MEMORY TYPE	memory access type (see special codes)
6	NUMBER OF FULL BLOCKS	number of full blocks transferred
7	NUMBER OF BYTES IN LAST BLOCK	number of byte in last data block
8	MEMORY OFFSET LOW	target memory starting address
9	MEMORY OFFSET HI	
10	MEMORY SEGMENT LO	
11	MEMORY SEGMENT HI	
12	ETX	end of text character
13	LRC	longitudinal redundancy check

Table B-1

All data values are expressed in hexadecimal notation.

The slave ID number is the same that appears in the enquiry sequence.

The Master ID number must be 01 hexadecimal.

The Read/Write byte identifies what type of operation is to be executed. A write operation will transfer data from the slave device to the master. The read operation is the opposite. It transfers data from the master to the slave. The following codes are used:

READ	01h
WRITE	00h

The memory access type specifier identifies specific areas of memory that the data transfer can take place in. This eases the burden of programming on the programmer in many cases.

The elementary code '00h' is the general purpose memory access code. This code enabled the programmer to read or write any byte in the 8086 memory space. This makes the command a very powerful and also a potentially hazardous tool to work with. CRS technical staff should be consulted when this command is used.

A list of the special codes available appears in a later section.

The next two byte in the header identify the amount of data to be transferred. This information is considered as the number of full data blocks to be transferred, and the number of bytes which remain in the last data block. The last data block cannot have 0 bytes in it. A full block contains 128 bytes of data. If the number of bytes to be transferred is an exact multiple of 128, then the last block will contain 128 bytes (80 hex). The logic to determine the number of blocks and the number of bytes in the last block is as follows;

```
let N be the total number of bytes to be transferred,  
let B be the number of full blocks to be transferred,  
let n be the number of bytes in the last block;
```

```
B = N / 128  
n = N mod 128      ; (ie. the remainder of the division N/128)  
if n = 0 then do  
    n = 128  
    B = B - 1  
endif
```

The memory address identified the starting address of the data transfer operation. It defines (in Intel segment/offset format) where in the robot memory the data will be transferred to or from during this operation.

Header Transfer Error Detection

The LRC byte is the longitudinal redundancy check character. It is the sum of bytes 2 through 11 inclusive of the header block. The slave device sums the values of all received characters and will accept the header block only if it reads an LRC that is the same as its own computed value. If the LRC's do not match, then the header block is not accepted, and the slave will issue a **NAK** character in response so that the master control will know to re-try the header block transfer. The master can only have three re-try attempts before the slave device issues an error, and aborts the communication cycle. If the header block is accepted, the slave device will issue an **ACK** character as a response. The slave also expects the correct control codes **SOH** and **ETX** during the transmission, and any deviation from this pattern will cause an error, and the communication will be aborted. Timeout checks are made between the end of the enquiry sequence, and the acceptance of the SOH character, and also the length of time required between sending an SOH and ETX character.

B-6 DATA BLOCK TRANSMISSION SEQUENCE

The data block transfer sequence is determined by the format specified in the header block.

In a data write command, the slave will begin to write the specified data blocks quickly after it issues an ACK in response to receiving the header block. It expects ACK characters from the master after each block is transmitted. Receiving a NAK will provoke a retry of the previous block. After the last block is sent and acknowledged, the slave will issue an EOT. It expects to receive a final EOT from the master, and then will close down the communication. The data blocks are configured in the following manner;

STX + [DATA BYTE #0] + ,.....,+ [DATA BYTE 127] + ETB + LRC

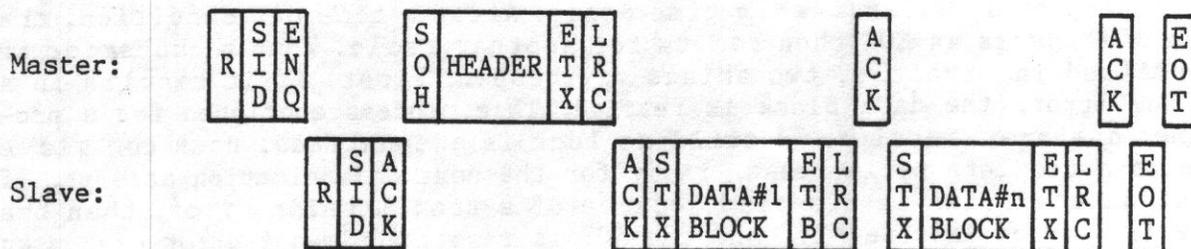
for the full data blocks, and

STX + [DATA BYTE #0] + ,.....,+ [DATA BYTE #n-1] + ETX + LRC

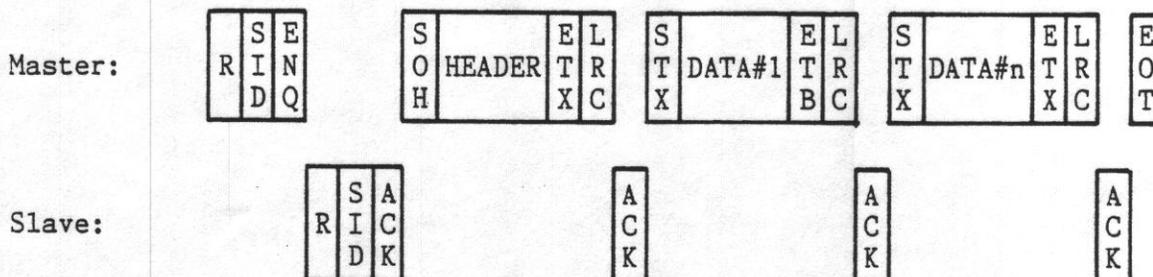
for the last data block.

The last data block uses an ETX character instead of an ETB character in order to signal the very end of the transferred data.

The following diagram illustrates the flow of data between the master and slave devices during a separate read and write cycle.



Similarly, in a read command, the slave will expect blocks of data after the header is acknowledged. It will issue a NAK if it detects an error in transmission. The master will issue an EOT after the last data block is read and acknowledged.



Data Transfer Error Detection

As in the header block transfer, the LRC byte is the sum of all data bytes in the block. Control codes STX, ETX or ETB are not included in the summation. An erroneous LRC check will cause the receiving end to issue a NAK character, which will force a re-try of the entire block. Only three re-tries are permitted, whereupon the receiving end should issue an EOT and abort the sequence. Timeout checks are made for reception of an STX after a complete header transfer, or a complete previous data block transfer, and also between accepting the opening STX of a data block transfer, and reading the ETX or ETB character. Once a data block has been accepted by the ACI in a read operation, the data block is transferred to the target memory address.

B-7 EOT SEQUENCE

The EOT is the final sequence, and is shown in the timing charts above. In the communication, the master ALWAYS has the last word, and the master ALWAYS has the ability to abort the communication. The slave will never knowingly abort the communication once it is started.

B-8 ERROR CHECKING AND RECOVERY

The ACI can handle a range of different communication errors. The two broad classifications of errors are transmission inaccuracies, and character time outs. Character time outs are used when a pending communication cycle has been interrupted by an unforgivable time delay. After a time out condition, the slave ACI issues an EOT then resets for another cycle. When a character is transmitted incorrectly, two things may happen. First, if it results in a checksum error, the data block is retried. This process continues for a programmed number of retries. If still no luck is encountered, then the slave executes a complete cycle reset, ready for the next communication attempt. If a control character is mis-read because of a transmission error, then the cycle is aborted immediately, and the ACI is reset. A timeout error causes an immediate reset of the ACI.

Error Number	Error Description, Cause, and recovery
1	Not used
2	No SOH character found to lead off the header sequence. The SOH character must be the first character of the header. Any other character will cause the ACI to terminate, and the communication cycle will be ended. The ACI will release an EOT character.
3	The ACI encountered a software error when reading in the header block. This is an internal software failure. Contact your CRS Plus representative. The ACI will be reset after this error.
4	A header retry failure was encountered. This means that the ACI tried four times to obtain the header, and was unable to do so due to repeated LRC failures. Usually, this is a sign of a bad interface wiring, or incorrect master control software. If you are using CRS Plus interface software for your host computer, contact your CRS Plus representative.
5	No ETX, or ETB was read at the end of a data block transfer. Remember that the last data block must have ended with an ETX character. All other data blocks end with an ETB character. The ACI will immediately reset after detecting this error.
6	Not used
7	Not used
8	Data block read retry failure. The ACI terminated due to repeated LRC check failures when reading a data block from the Master.
9	Character timeout. The ACI expected a character, and did not receive one in the allotted time. The ACI will be reset. Check that the Master control did not hang up part way through a transmission, or that the correct number of characters were sent.
10	The ACI read a character other than an STX at the start of a data block. The ACI will be reset.
11	Data block write retry failure. The ACI was unable to deliver a data block to the Master without getting a NAK response four consecutive times

Table B - 2 ACI Error Codes

Error Number	Error Description, Cause, and recovery
12	A character other than an ACK or NAK was read in response to a data block write. This error can occur on a spurious noise injection into the line, but is a rare error type. If this error persists, then the Master could be responding with an incorrect code. Contact your CRS Plus representative. The ACI will be reset.
13	Timeout error on transmission of a character. The ACI was not able to transmit a character in the allotted time. This timeout would have been caused by a failure in the DART chip (see hardware description) to present a transmit ready signal. The ACI will be reset after this error. Contact CRS Plus if the problem persists.
14	Bad target Id match in header block. The header block must contain the same slave device number that the enquiry sequence had, or this error will be generated. The ACI will be reset.
15	Not used
16	Bad character received when EOT expected. The ACI expects the Master to close all communication with an EOT character. This error is generated if this does not happen, and another character is received instead. Since the communication is already finished by this point anyway, the ACI will be reset.
17	A character other than an ETX was received at the end of a header block. This character must be delivered to the ACI immediately after the LRC character is sent.
18	Timeout on receiving the header SOH character. The Master control, after establishing communications with the slave during the enquiry sequence, has a limited time to send the following header block. See the table of timeout values. A timeout error always causes the ACI to reset.
19	Timeout of receiving the header ETX character. This timeout measures the time it takes to complete the transmission of the header block. Once the SOH has been received, the Master has only a limited time to transmit the whole header block. The ACI will be reset after this error.
20	Timeout on receiving the data block STX character. When the header block has been completed, and the Master has programmed a read cycle for the slave, the Master must deliver the first data block within a time period. Failing to do so will reset the ACI.

Table B - 2 ACI Error Codes

Error Number	Error Description, Cause, and recovery
21	Timeout on data block ETB/ETX character. Similar to error #19, the master has only a limited time to send the entire data block before this error occurs. The ACI will be reset.
22	Timeout on receiving a data block ACK/NAK character. After the slave has transmitted the data block to the Master, the Master must respond with the ACK or NAK character within the preset time limit, or the ACI will reset automatically.
23	Timeout on receiving the final EOT. The slave expects the Master to close the communication within a specified time limit. Failure to do so will set this error, and the ACI will be reset.
24	Unexpected EOT character. This error will be set if any expected control character turns out to be an EOT. The Master can close off the communication prematurely by sending an EOT instead of the accepted control code at any time in the sequence.
25	Not used
26	Bad special read code. The read code provided in the header is not supported.
27	Bad special write code. The write code provided in the header is not supported.
28	Bad special memory access code. The memory access code provided in the header is not supported.

Table B - 2 ACI Error Codes (cont'd)

B-9 ABORTING A COMMUNICATION CYCLE

Sending an EOT at any time that a normal control character is to be issued will be interpreted by the slave as a signal to abort the communication. This function may be required at times where an abrupt termination of the link is essential, for instance when attempting to quickly service another slave.

B-10 INTERFACE REQUIREMENTS

The ACI has been tested at baud rates up to and including 2400 baud. Faster baud rates may be possible, but are not recommended. Handshaking should be disabled, and parity should be turned off. A data byte size of 8 bits is required, with two stop bits. See the RAPL CONFIG command for setting up serial channel 1 to service the ACI link.

B-11 PREPARING THE ROBOT CONTROLLER FOR A COMMUNICATION CYCLE

The robot controller must be set up to accept the communication. This communication is executed through serial port 1. When this is active, then any robot references to device #1 will not respond, since this protocol takes priority when it is activated.

B-12 ACI MONITOR COMMANDS

The programmer can access the ACI software through a set of commands available at the terminal. These commands will allow the programmer to monitor any communications to and from the controller. The programmer can also enable or disable the ACI interface.

The commands are only available through the monitor level.

@@RI:

To force an initialization of the ACI interface. Any communication in sequence will be aborted. This may cause a loss of synchronization between the slave unit and the master control.

@@RN:

To select a slave device number for the robot controller. This value can be any value between 1 and 127. The value of 0 is not permitted.

@@RS:

Display the current status of the ACI software. The number of retries, the number of communication failures and the number of successful cycles will be displayed in a continuous fashion on the terminal device. This is a useful debugging tool.

@@RE:

Enable the ACI software. This will dedicate the serial port #1 to the ACI software. No other normal serial input will be allowed on this channel. Output is still allowed, but it will cause problems if an ACI cycle is concurrently running.

@@RD:

The ACI interface will be disabled. The serial port #1 will now be considered as any other programmable serial channel.

@@RH:

The last header which was read in will be displayed. This will indicate the type of the last (or current) conversation.

Character	Hex Code	Decimal Code
STX	02	02
ETX	03	03
ETB	17	23
EOT	04	04
SOH	01	01
ENQ	05	05
ACK	06	06
NAK	15	21

Table B-3 ASCII control codes for the ACI

Event	Timeout (in character times ++)
Enquiry timeout Timeout used to test for characters received during the enquiry sequence.	300
SOH timeout Time permitted between the recognition of a successful enquiry sequence, and the acceptance of the header SOH character	300
Header ETX timeout Time permitted between acceptance of the header SOH character, and the detection of the header ETX character.	300
ACK for data block Time permitted for waiting for an ACK or NAK from the master after the transmission of a data block from the slave.	300
STX data block timeout Time permitted for waiting for the data block STX from the master after a successful acknowledgment of either the header block, or the the previous data block.	300
Data block ETX/ETB timeout Timeout permitted for the complete data block, from acceptance of the STX to the reception of either the ETB or ETX character.	300
Final EOT timeout Time permitted between the acknowledgement of the last data block read and the final EOT response, or the last data block write acknowledge from the master, and its response to the slaves final EOT.	300

Table B-4 ACI timeout values

++ Timeouts are given in character times so that the baud rate does not affect the timeout strategy.

B-13 SPECIAL ACI ACCESS CODES

This chapter deals with the special functions available with the ACI version 2 software from CRS Plus. The extra functions located here explain how the system programmer can interface to the robot control to obtain information that was previously very difficult to obtain.

The functions explained here will permit the programmer to access robot I/O ports, as well as position and command registers.

The special codes are separated into special write codes (codes 40h to 4Fh) and special read codes (20h to 2Fh). These codes are placed in the access type specifier in the header block.

Special Write Codes

With the exception for code 47h, it is not necessary to specify an address with special write commands. It is important, however to specify the amount of data to be transferred. This is important in all cases.

40h

This special write command will return the most relevant addresses within the robot software. The addresses which are returned will enable the master control to access user memory directly, thereby permitting program upload and downloading. The use of this function will provide master control software which can operate independently from the robot software version. The buffer of information which is returned is in the form of memory pointers (4 byte, segment, offset format). The lowest byte contains the least significant byte of the pointer. The addresses included in the buffer of information point to the items in the robot memory as described in table F-8, in Appendix F of the technical manual. The buffer of pointer information is the same as that used by the RAPL-BIOS facility, and is not reproduced here. With this special code, no address field need be specified, but since the pointer information consists of more than one full block, any portion of the data can be returned for use, starting with the first element.

41h

Send robot error codes. This special write code will send four bytes to the master indicating the health of the robot control. The buffer of four bytes includes, in this order:

1	AlarmStatus	00	indicates no alarmcondition
		01	indicates that an alarm exists
2	RAPL error code	NN	See the RAPL error manual
3	ACI error	00	No ACI error in effect
		01	ACI error is in effect
4	ACI error code	NN	See the ACI error list. This is the last ACI error detected.

42h

This command will force a write of the user input port images as captured by the RAPL operating system. The memory address in the header block need not be initialized, but the byte count must be specified. That is, it should contain a value of 8. See the technical description of the robot I/O space for a description of the inputs that can be read.

43h

This command will force a write of the user output port images as created by the RAPL operating system. The memory address in the header block need not be initialized, but the byte count must be specified. That is, it should contain a value of 8. See the technical description of the robot I/O space for a description of the inputs that can be read.

44h

This command will cause a write to the host computer of the position command data in the robot controller. The position command data exists as an array of 8 double integer values in memory. No address need be specified in the header block, but a data length of 32 bytes or less should be specified.

45h

This command will cause a write to the host computer of the actual position data in the robot controller. The actual position data exists as an array of 8 double integer values in memory. No address need be specified in the header block, but a data length of 32 bytes or less should be specified.

46h

This command will cause a write to the host computer of the end point of path data in the robot controller. The end point position data exists as an array of 8 double integer values in memory. No address need be specified in the header block, but a data length of 32 bytes or less should be specified.

47h

This command will transfer 8086 I/O input port values to the host computer. The I/O address is specified by the offset portion of the data address field in the header block. The segment field is not used. The number of bytes specified will result in a read of all ports from the specified address. This is a special command in that only one partial block of data can be transferred. The number of bytes in the last block will then correspond to the number of I/O ports to be scanned.

Special Read Codes

Special read codes allow the programmer to load specific areas of the robot memory with data.

20h

This special read command will load up to 128 bytes into the active command input buffer. In this way, an external computer can simulate a data terminal entry. Only text characters will provide a valid response. Any attempt to enter control codes as you would at the interactive level with the terminal will produce a RAPL error. No echoing will be noticed, since all normal echoes will go to the terminal.

Appendix B: Reprint of Technical Manual, Appendix G

```
0001 /*
0002 -----
0003
0004 Module : ACI00.C
0005 -----
0006
0007 Description :
0008 -----
0009 This module will provide all of the ACI protocol support routines. The
0010 requirements for the ACI implementation can be divided into four procedure
0011 categories:
0012 i) Hardware Implementation, (the list below specifies which routines must
0013 be supplied by the programmer for different implementations). These
0014 procedures are found external to this one.
0015 ii) Utility routines,
0016 iii) Protocol level. These are the procedures which control and enforce the
0017 ACI protocol standard,
0018 iv) Application Interface. These procedures provide quick and easy integration
0019 of the ACI library to an existing application software package
0020
0021 Type i) procedures are defined in IBM00.C, or must be provided by the system
0022 programmer.
0023
0024
0025 Additional Information
0026 -----
0027 The RPL Technical reference manual, notably Appendix B discusses the ACI
0028 protocol in detail. This information is not reproduced here.
0029
0030
0031 Global Procedures Defined :
0032 -----
0033
0034 Utilities
0035 -----
0036
0037 void aci_putstr( str , n )
0038 char str[];
0039 int n;
0040 Send a string of characters out. The string is pointed to by 'str', and the
0041 string is 'n' bytes long.
0042
0043 void aci_strin( tempstr,n )
0044 char tempstr[];
0045 int n;
0046 Read a string of characters in from the serial interface. The number of characters
0047 to be expected is 'n', while the destination character array is pointed to
0048 by 'tempstr'.
0049
0050 int lobyte(i)
0051 int i;
0052 Will return the low byte of the integer.
0053
0054 hbyte(i)
```

APPENDIX G - RAPL CONTROL PARAMETER LIST

TABLE OF CONTENTS

G-1 INTRODUCTION	2
G-2 NOTE FOR RAPL 5.00 AND LATER VERSIONS	2
G-3 RAPL PARAMETER POINTER LIST	2
G-4 RAPL USER MEMORY ALLOCATION	10
Program Buffer	10
Program Table	10
Location Table	11
Variable Table	11
Trigger Table	12
Continuous Path Memory Allocation	12

G-1 INTRODUCTION

RAPL software provides a list of pointers so that the user can access system parameters in RAPL using the ACI interface. See Appendix B for a complete description of the ACI interface. By providing this list of pointers, the programmer can be isolated from version changes in the RAPL software that could result in parameter address changes. This parameter list is accessible through special ACI write command 40 hex. A pointer to this list of pointers is also available at interrupt vector 60 decimal. The list of items is as follows.

Data found at the pointers indicated by this list are identified by type in the left hand margin: B for bytes, I for integers W for words, DI for long integers, R for real numbers, P for pointers and DW for long words comprise the data type specification. Arrays of types are shown with the appropriate array length. Remember, the items in the list are pointers. Some of these may point to the address of pointers (P type data) in the robot memory which then finally point to the data in question. A pointer to this list of pointers resides in interrupt vector 60 decimal.

G-2 NOTE FOR RAPL 5.00 AND LATER VERSIONS

Some system parameters have been replaced in this version. These obsolete parameters are indicated below. When the pointer list is read in the robot memory, these entries will appear as zero (NULL) pointers.

G-3 RAPL PARAMETER POINTER LIST

Data Type	Offset in List	Item Description (This pointer points to..)
P	0	the work space pointer - Pointer to the RAPL user space. Also the pointer to the RAPL program buffer. <code>PROG_BUFF_PTR</code>
W	1	the Program buffer size - the word register which tells us how many bytes of program storage has been allocated. <code>PROG_BUFF_SIZE</code>
W	2	the Program buffer count register - the word register which tells us how many bytes of program storage has been used. <code>PROG_BUFF_CNT</code>
P	3	the Program table pointer - the Pointer to the location of the program table in memory. <code>PROG_TABLE_PTR</code>
W	4	the program table size - the Word register which tells us how long the program table is in number of total program entry locations. <code>PROG_TABLE_SIZE</code>
P	5	the Variable table pointer - the Pointer to the variable table. <code>VAR_TABLE_PTR</code>
W	6	the Variable table size. Word register which tells us how big the variable table is in the total number of entries. <code>VAR_TABLE_SIZE</code>
P	7	the Location table pointer. Pointer to the location table. <code>LOC_TABLE_PTR</code>
W	8	the Location table size. Word register which tells us how long the location table is in the total number of table entries. <code>LOC_TABLE_SIZE</code>

Table G-1

Data Type	Offset in List	Item Description (This pointer points to..)
DI*8	9	the Calibration register pointer. Pointer to an array of 8 double integers which contain the robot calibration position. CALIBRATION(8)
B	10	the Calibrate position checksum byte. The byte addition of all valid calibration data for the 5 robot axes only. CALIBRATE_CHECKSUM
B	11	the 'robot is calibrated' check byte which is set after a calibration procedure has been run. ROBOT_IS_CALIBRATED
B*8	12	the Axis 'Begin Motion' array. An array of 8 bytes which signals each axis command generator to start a new motion. It is reset by the command generator when the path starts. BEGIN_MOTION *** NOTE: DISCONTINUED FOR RAPL VERSION 5.00 AND LATER.
B*8	13	the Axis 'Done' array. An array of 8 bytes which signals when each of the axes has finished a motion. Set by the command generator. DONE *** NOTE: DISCONTINUED FOR RAPL VERSION 5.00 AND LATER.
DI*8	14	the Actual Position array. An array of 8 double integers which are the absolute position registers of the robot feedback sub-system. Every 4 milliseconds, these values are updated with the incremental positions determined by the motor encoders. POSITION
DI*8	15	the Position Command array. An array of 8 double integers which represent the absolute position command to the robot motors. Any path generation algorithm can create these values, and the servo loop function will compare these registers to the actual position registers and will create the appropriate commands. POSITION_COMMAND
DI*8	16	the End point registers. An array of 8 double integers which define the end point of the current motion command in motor coordinates. If the robot is not moving, then these values equal the position command registers. END_POINT(8)
B	17	the 'Move type' parameter. This flag is set by the path generation selection. MOVE_TYPE The value can be: 1 - For joint interpolated moves, 2 - reserved 4 - manual mode. 5 - continuous path mode (and straight line moves)
B	18	the Number of active axes. Byte value with value from 1 to 8. NUMBER_OF_AXES
B*8	19	the Limp command array. Array of 8 bytes which selects the limp mode for each motor. In the limp mode, the servo loop function automatically overrides the command generation function, and will force the position command to be the same as the actual position registers. This supplies the motors with 0 voltage. LIMP(8) *** NOTE: DISCONTINUED FOR RAPL VERSION 5.00 AND LATER.

Table G-1

Data Type	Offset in List	Item Description (This pointer points to..)
DW	20	the Real time ticker. A double word quantity that is incremented each I/O update cycle (approximately 30 milliseconds). The value of this counter is zeroed only at a teach start situation, so it can accurately keep track of the total time that the controller has been switched on. REAL_TIME
B	21	the Hold request flag byte. A byte value that, when set to 1, will stop a RAPL path generation, and will cause all DONE flags to register a complete path. The end point registers of the path will be updated to the commanded position of the axes when all axes have been brought to a stop. In joint interpolated mode, the axes are decelerated to a stop. In straight line, the axes are stopped immediately. The hold request is reset when a new move is commanded. HOLD_REQUEST
B*8	22	the Axis Lock out array. Pointer to an array of 8 bytes which will selectively lockout individual motors from any subsequent motion commands. LOCKOUT(8) *** NOTE: DISCONTINUED FOR RAPL VERSION 5.00 AND LATER.
W	23	the Millisecond counter. A word quantity which is used to derive a millisecond delay from the system I/O clock update. Used to time the DELAY function in RAPL. MILLISECOND_COUNTER
R*8	24	the Acceleration array. An array of 8 reals which determine the acceleration of each motor during a joint interpolated motion. ACCELERATION(8)
R*8	25	the Transmission Ratio array. An array of 8 real numbers which equates the measurement units of the joint or motor with the corresponding motor pulse resolution. It is dangerous to change the first five values in this array, as it will change the robot transformation. The @XRATIO command changes the other three.
B	26	the Alarm Number. A byte value that contains the last RAPL error code to be generated. ALARM_NUMBER
W*8	27	the Rotary Resolution array. An array of 8 integers which specifies the motor encoder resolution in encoder lines per motor revolution. ROTARY_RESOLUTION(8)
R*6	28	the Tool Transform - an array of 6 real numbers that specify the current tool transform value. TOOL_TRANSFORM

Table G-1 (Cont'd)

Data Type	Offset in List	Item Description (This pointer points to..)
Struct	29	<p>the Input block structure - the RAPL character input buffer area. The input buffers are specified in the following format:</p> <pre> input_block(3) structure (bufsize byte, in_ptr byte, out_ptr byte, buffer(128) byte) </pre> <p>The first two structure records keep track of data input from the two serial ports. The third is used as an intermediate buffer when executing from a program. 'bufsize' determines the size of the input ring buffer, up to 128 bytes. 'in_ptr' points to the next array element available for serial input, and 'out_ptr' marks the next character that can be released from the buffer, to be used by RAPL.</p>
B	30	<p>the Alarm status. A boolean byte value which specifies whether or not a RAPL error has occurred. The byte is reset when a new command is attempted. ALARM_STATUS</p>
R*64	31	<p>the Motor to Joint transform matrix. An 8 by 8 real matrix which transforms the motor coordinates into the joint coordinates. Since the robot structure is not mechanically de-coupled, several motors may have to move in order to provide a single joint motion. In a matrix equation, $\{j\} = [J]\{m\}$, where $\{j\}$ is the joint row matrix, and $\{m\}$ is the joint motor matrix.</p>
R*64	32	<p>Joint to motor transformation matrix. An 8 by 8 real matrix which determines motor coordinates from joint coordinates. It is the matrix inverse of item #31 above. In a matrix equation, $\{m\} = [JJ]\{j\}$, where $\{j\}$ is the joint row matrix, and $\{m\}$ is the joint motor matrix. The JJ matrix is the inverse of the J matrix identified by item #31.</p>
W	33	<p>the Work space size. A word value which specifies the total user area available. The user area can be represented as an array in PL/m-86 as follows.</p> <pre>(User_Area based work_space_ptr)(work_space_size) byte;</pre>
B*3	34	<p>the Extra axes calibrate checksums. A 3 byte array which contains the checksums for the calibrate position of each extra axis. XCALIBRATE_CHECKSUMS(3)</p>
W	35	<p>the Reserved work space size. This word contains the number of bytes which are to be set aside from the routine RAPL user memory. see section F-9 for a description of the RAPL user space. This memory area can then be used for 8086 machine code programs that can be loaded in from the terminal, or through the ACI. This area is also used for CTPATH data. RESERVED_WORK_SPACE_SIZE</p>
B*9	36	<p>the Digital Input Image buffer. A buffer of 72 bits which map the robot input space. See figure F-5. DIGITAL_INPUT</p>

Table G-1 (Cont'd)

Data Type	Offset in List	Item Description (This pointer points to..)
B*7	37	the Digital Output Image buffer. A buffer of 56 bits which map the robot output space. Since the robot digital output ports cannot be read, then it is up to the software to remember what was issued to the output port before updating single bits of it, so that the remaining bits can be left as they were. See figure F-4. DIGITAL_OUTPUT
B	38	the Default character I/O device. The default character device is either 0 or 1 or 2. 0 and 1 represent the two serial channels. 2 represents a user program which is being used for character input. The default device is 2 only when using the controller without a terminal in the AUTOSTART mode. When a program is executing, the default device represents the serial channel that is to be used for any user input and output. DEFAULT_DEVICE
B	39	the Device Select. Identifies the current device being used for character input and output. DEVICE_SELECT
B	40	the Syntax Help function. This byte boolean value determines whether or not the Syntax building function will be activated or not. HELP_ON
B*8	41	the teach name template. An array of 8 bytes which contain the current teach name. Only the first 5 characters are used when the final point name is formulated. The rest is padded with underscore characters. TEACH_NAME_TEMPLATE
B	42	the teach name count. The byte value which is encoded into the final three character positions of the teach name when the point is finally stored. TEACH_NAME_COUNT
B	43	the Program pause flag. When a program is executing, setting this byte to a '1' value will stop the program after the next line is decoded. PROGRAM_PAUSE
B	44	the homing complete flag. This flag is set when the robot has been homed after a power up. Do not confuse this with item #11, which indicates whether or not the robot has been calibrated. This flag must be set in order that all move functions that access points will work. ROBOT_IS_HOMED
B*128	45	the string buffer. The string buffer is a buffer 128 bytes long, that is divided evenly into 4 32 byte areas. These correspond to the 4 string variables that are allowed by RAPL version 3.50 or later. STRING_BUFFER
B	46	the loss of feedback/collision detection byte. This byte indicates the status of this function. A boolean true means that the function is turned on. LOFB_CHECK
B	47	the flag indicating whether or not a program is completed or not. PROGRAM_COMPLETED
W	48	the word counter that indicates the number of repetitions that the current program has completed. PROGRAM_REPETITIONS

Table G-1 (Cont'd)

Data Type	Offset in List	Item Description (This pointer points to..)
W	49	the word register which indicates the number of repetitions of the program that have been requested. PROGRAM_REPEAT_LIMIT
B	50	the flag which indicates that an infinite looping of the current program has been requested. PROGRAM_LOOP_FOREVER
R*250	51	the 1000 byte buffer which contains the data for the current straight line move. Also used to contain the path parameters for a CPATH. STRAIGHT_LINE_BUFFER
B	52	the flag which indicates whether or not the trigger table has been enabled. TRIGGER_TABLE
B	53	the flag which indicates whether or not editing is permitted. EDIT_ENABLE
I	54	integer containing the global robot speed value. ROBOT_VELOCITY
R*6	55	the first element of the robot base shift registers. The shift registers are arranged in this order: X_BASE_SHIFT, Y_BASE_SHIFT, Z_BASE_SHIFT, O_BASE_SHIFT, COS_O_BASE_SHIFT, SIN_O_BASE_SHIFT
B	56	the flag which indicates whether or not a continuous path has been interrupted with the HALT command, typically in an ONSIG situation. CPATH_INTERRUPTED
DI*8	57	the 8 double integer registers used to save the path position were the robot was interrupted from its continuous path. HOLD_POSITION
B	58	the register which describes the configuration of the robot arm for all coordinate transformations. CONFIG
P	59	the pointer to the knots array for the current CPATH or straight line move. P_KNOTS
P	60	the pointer to the time array for the current CPATH or straight line move. P_TIME
P	61	the pointer to the acceleration array for the current CPATH or straight line move. P_ACCEL
P	62	the pointer to the temporary array for the current CPATH or straight line move. P_TEMP
P	63	the pointer to the knots array for the current CTPATH or straight line move. TP_P_KNOTS
P	64	the pointer to the time array for the current CPATH or straight line move. TP_P_TIME
P	65	the pointer to the acceleration array for the current CPATH or straight line move. TP_P_ACCEL
P	66	the pointer to the temporary array for the current CPATH or straight line move. TP_P_TEMP
R	67	the TIMER which increments as the continuous path proceeds. This timer registers is compared to the knot timer array in order to determine which path segment the path is currently in.

Table G-1 (Cont'd)

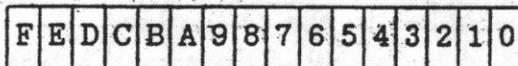
Data Type	Offset in List	Item Description (This pointer points to..)
W	68	the <u>PATH_SEG</u> which determines which path segment the path is currently in.
W	69	the number of knots that are in the CTPATH. <u>TP_N_KNOTS</u>
W	70	the number of axes which are included in the current CTPATH. <u>TP_N_AXES</u>
W	71	the number of knots that are in the last CPATH. <u>N_KNOTS</u>
W	72	the number of axes that are included in the last CPATH. <u>N_AXES</u>
B	73	the flag which indicates whether a CTPATH is loaded and ready to go. <u>PATH_LOADED</u>
W	74	the register which determines the maximum number of knots which will be used in the calculation of all straight line moves. <u>ST_PATH_KNOTS</u>

NOTE: THE FOLLOWING ARE AVAILABLE FOR RAEL VERSION 5.00 AND LATER

B*8	75	8 character buffer which identifies the first location in a CTPATH, used to send the robot under joint interpolated mode to the start of the path. <u>CPATH_START_NAME</u>
B	76	Enable NULL positioning mode. In this mode, the finish criterion is not complete until each axis has moved to within a preset tolerance band of the end point. <u>NULL_ON</u>
B*8	77	The tolerance band for NULL positioning is set for each axis. Byte values allow for 127 pulse positioning error. <u>NULL_TOLERANCE</u>
B	78	Enable the trigger events during the next CPATH, CTPATH. <u>TRIGGER_ENABLE</u>
B	79	Enable outputs to be issued. Disabling the outputs is often useful when debugging a program without wanting to switch external devices. <u>OUTPUT_ENABLE</u>
B	80	Enables the ONSIG condition without changing the setup status of the command. <u>ONSIG_ENABLE</u>
B	81	Enable the arm power watchdog. Disabling will mean that the arm power will be switched off within 16 milliseconds. <u>ARM_ENABLE</u>
B	82	Enable the TRACE mode display. <u>TRACE_ENABLE</u>
B	83	Enable the FLASH function on the teach pendant LED. <u>FLASH_ENABLE</u>
B	84	Enable the check for loss of arm power. <u>ARM_POWER_CHECK_ENABLE</u>
B	85	Enable the teach mode with the prescribed template name and count number. <u>TEACH_ENABLE</u>
B	86	Enable the manual mode, under the currently active manual mode type. <u>MANUAL_MODE_ENABLE</u>
B	87	Enable the cartesian velocity continuous path control. This flag will ensure constant cartesian velocity control. <u>CARTESIAN_VELOCITY_ENABLE</u>
B	88	Enable the SLEW velocity mode for joint interpolation. This mode is faster than SPLINE mode, where the velocity shape has a parabolic characteristic. <u>TRAPEZOIDAL_VELOCITY_ENABLE</u>

Table G-1 (Cont'd)

Data Type	Offset in List	Item Description (This pointer points to..)
B	89	Stop any current motion and put the command generator in a suspended mode. FEED_HOLD_ENABLE
P	90	Pointer to the available user memory. This is a pointer to the reserved space. MEMORY
W*8	91	Array of 8 words which defines the status of the controller axes. This array is organized as a bit array. It replaces the old byte flag arrays: DONE , BEGIN_MOTION , LIMP , LOCKOUT , AXIS_STATUS . The bit allocation of this word array is shown below;



- └─ **AXIS OK.** This bit will be set to 1 when the axis card is healthy.
- └─ **LIMP.** bit will be set to '1' when the axis is under the limp mode.
- └─ **LOCKOUT.** When this bit is 1, the axis will be excluded from any future move commands.
- └─ **AXIS ERROR.** When this bit is set, the axis currently under test has an error condition.
- └─ **MOTOR FAULT.** When this bit is set, the axis card has detected a motor fault condition, and this condition has been recognized by the RAPL operating system
- └─ **HOMED.** When this bit is 1, the axis has been homed.
- └─ **CAL.** When this bit is set to 1, the axis has been calibrated.
- └─ **BEGIN MOTION.** When this bit is set, the RAPL command interpreter has requested a path start. This bit will be reset when the path does start.
- └─ **DONE.** When this bit is set to '1', there is no path in progress on the current axis.

NOTE: All unused bits are reserved by the RAPL operating system. Also, modification of these registers is strictly not recommended, as they are constantly updated by the RAPL communication software with the smart axis cards. Use these registers as status indicators only.

Table G-1 (Cont'd)

G-4 RAPL USER MEMORY ALLOCATION

Please note that addresses of pointers to items in the robot user memory can be determined from the pointer list described in Table G-1. All references to "the table" mean Table G-1.

The RAPL user memory is partitioned into 5 areas. These are listed according to memory order. User memory is pointed to by the `PROG_BUFF_PTR` parameter, item #0 in the table. Since this is the bottom of user memory, this pointer is also referred to as the `work_space_ptr` parameter.

The reserved memory area is the first item in the user memory. It can be reserved for any purpose so that RAPL program, location and data operations cannot affect it. If the expanded memory option is installed in the controller, This memory can be used to store path data generated by the "CTPATH" command. The `work_space_ptr` parameter is adjusted to point to the next byte beyond this area. All other RAPL user memory items are addressed relative to the `work_space_ptr` parameter.

Program Buffer

The program buffer, which is a contiguous array of bytes is stored next. Each program is stored immediately after one another. A '\$' character separates one program from the next. The program buffer is addressed by the `work_space_ptr` parameter (#0) since it appears first in memory.

The program table is next. It can best be described by the following 'C' structure reference:

Program Table

```
struct p_table
{
    char name[8];
    int index;
    int checksum;
} prog_table_ptr[prog_table_size];
```

And the pointer to the program table (item #3 in the table) can be calculated to be:

```
prog_table_ptr = &work_space_ptr[prog_buffer_size];
```

Each entry into the program table takes 12 bytes. The `PROG_PTR` element defines the starting point in the program buffer for that particular program. The actual location of the program is then at:

```
prog_address = &work_space_ptr[prog_table_ptr[i].index];
```

The checksum byte is the addition of all the program bytes up to the final '\$' character. The checksum is only a byte length addition, so the upper byte of the checksum word is undefined and should be set to '0'.

Location Table

The location table is next. It can be described by the following 'C' structure reference:

```
struct l_table
{
  char name[8]; 8 bytes
  float data[8]; + 32 bytes = 42 bytes
  int checksum; + 2 bytes
} loc_table_ptr[loc_table_size];
```

For a precision point, the location table would be described as:

```
struct l_table
{
  char name[8];
  long pp_data[8];
  int checksum;
} loc_table_ptr[loc_table_size];
```

Thus any entry in the location table requires 42 bytes of data: an 8-byte name, 8 real or long integer fields (4 bytes each) and a checksum word. The checksum is the addition of all the bytes in the particular location. The checksum is only a byte length addition, so as with the program table, the upper byte of the checksum word is undefined.

The location table pointer can be calculated to be:

```
loc_table_ptr = &prog_table_ptr[prog_table_size];
```

Variable Table

The variable table is stored next. It can best be described by the following 'C' structure reference:

```
struct v_table
{
  char name[8];
  float data;
  int checksum;
} var_table_ptr[var_table_size];
```

An entry in the variable table requires 14 bytes: an 8 byte field for the name, a 4 byte real value, and the two checksum bytes. As with the program and location table entries, the checksum is the addition of all the bytes in the particular location. The checksum is only a byte length addition, so the upper byte of the checksum word is undefined.

The position of the variable table can be calculated as follows:

```
var_table_ptr = &loc_table_ptr[loc_table_size];
```

Trigger Table

The trigger table is an item in the RAPL parameter memory. It is a structure of 8 elements called "triggers" which define programmed output events which will occur as the robot arm moves through locations (knots) during a CPATH or a CTPATH motion. The trigger table saves the location name of the event, as well as the output number and desired state of the output at the event. If the output number field is a zero, then no trigger event is loaded. A negative trigger number indicates that the selected output will be triggered to a LOW state. A positive number will trigger the output to a HI state. Valid output numbers range from 1 to 40. The trigger table can be defined as the following structure:

```
struct trig_table
{
    char name[8];
    int output_number_and_state;
} trig_table[8];
```

Thus an entry in the trigger table takes 10 bytes: the location name field consisting of an 8-byte entry and the integer state value. No checksum is provided as this data is not retained when the robot controller power is removed.

The data in the trigger table is used in execution of the CTPATH or CPATH commands. As the path data is generated, the trigger table is scanned. If a location is called for which a valid entry exists in the table, an entry is made in the path data for that knot.

During execution of the path, RAPL scans two word values at each knot. Each contains a map of the lower 16 user output line numbers. A "1" in a bit position in the first word represents an output whose state changes at that knot. The second word contains the desired state of that output after the change. The change is made at I/O scan rates so that the state will change within approximately 40 milliseconds from the time the commanded position of the robot reaches the knot.

Continuous Path Memory Allocation

The CTPATH command utilizes the reserved memory to store most of the path information. The remainder of information is saved in RAPL parameters pointed to by: TP_N_AXES, TP_N_KNOTS, TP_P_KNOTS, TP_P_TIME, TP_P_ACCEL, TP_P_TEMP, PATH_LOADED. These parameter pointers are shown in Table G-1 (items #63 to #66, #69 and #70). A more detailed description of these parameters can be found in the detailed discussion of the CPATH control strategy in the RAPL manual.

The first step in accessing the CPATH data is to determine the pointer to the reserved memory. For RAPL version 5.00 and later, this item is #90 in the pointer list. This pointer is not immediately accessible by any item in the pointer list for RAPL version 4.99 and earlier, but a simple operation can derive it, since it is accessible by taking the pointer to the workspace (item #0) and by

subtracting the reserved work space size (item #35) which would act as a negative offset from the start of the user memory. The following code extraction, written in 'C,' will do just that;

```
char *reserved_ptr, *work_space_ptr;
int reserved_space_size;
int *tp_n_knots, *tp_n_axes;

reserved_ptr = &work_space_ptr[-reserved_space_size];
```

Once the starting address to the data has been determined, the data elements for continuous path are stored in the following order;

```
float tp_pknots[n_axes][n_knots];
```

The knots array is located at the start of reserved memory. The knots array is actually a two dimensional array, bounded by the number of knots and the number of axes as the two dimensions. The total array size is therefore; $n1 = \#knots * \#axes * 4$, in bytes.

```
float tp_ptime[n_knots]
```

The time array is a float array that marks the elapsed path time at each knot.

```
float tp_paccel[n_axes][n_knots]
```

The acceleration array contains real numbers marking the acceleration of each axis at each knot position.

```
float tp_ptemp[n_knots]
```

This array is used for temporary storage during the calculation of the path parameters. It is then used to store the trigger setup information for the path. When storing the trigger information, the format of the data are two 16 bit word masks for each knot. The lower order word contains the bits which determine the outputs that are to be controlled at that particular knot. A 1 bit will indicate that the corresponding output will be controlled. The higher order word determines the state to which the designated outputs will be placed. A 0 bit will turn an output on, and a 1 will turn it off.

With this information, all of the pointers can be formulated, as the following code extraction demonstrates.

```
float *tp_pknots, *tp_paccel, tp_ptemp;

tp_pknots = reserved_ptr;
tp_ptime = &(tp_pknots[n_knots][n_axes]);
tp_paccel = &(tp_ptime[n_knots]);
tp_ptemp = &(tp_paccel[n_knots][n_axes]);
```

When it is time to execute a CPATH, using the GOPATH command, the pointer values and all of the related parameters are loaded into the working registers (items #236 to #248, #284 and #288) if the path loaded flag indicates that there is path data available.

ACI Library Procedures:

The following is a list of library routines included in the TURBO PASCAL ACI library. All data types and global variables requiring definition are also included. A simple example routine is also included with the listings.

data_transfer (slave_device: byte;
MemType: byte;
MemOfs, MemSeg: integer;
AccessType: char;
bytes: integer);

This routine coordinates a data transfer. It is the basic call made for any transfer of data using the ACI. Depending on whether a read or write is requested, it calls one of the following routines: data_read_from_slave or data_out. Each of these two routines call the Init_comm, header and send_header routines to initiate the communication. All of these routines call upon the lower level communications procedures listed here.

data_read_from_slave (n_bytes: integer);
Coordinate the transfer of data from slave.

data_out (n_bytes: integer);
Coordinate the transfer of data to the slave.

Init_comm (slave_no: byte);
Initiate communications with a slave. Calls the ENQout procedure.

ENQout (slave_no: byte);
Send an enquiry string.

header (slave_no, MemType: byte;
MemOfs, MemSeg: integer;
RorW: char;
message_length: integer);

Generate the Header Block in preparation for transmission.

send_header (slave, Memtype: byte;
MemOfs, MemSeg: integer;
RW: char;
data_length: integer);

Send the Header Block. Calls the previous routine.

error_state (err: byte);
Set error status.

turn_on_interrupt;
Turn on the serial interrupt state.

turn_off_interrupt;
Turn off the serial interrupt state.

clean_aux:

Clean the input buffer and support.

send_aux

(cha: char);

Send a character to the auxiliary port.

int_srv:

Service the serial on an interrupt basis.

init_8250

(rate: integer);

Set the baud rate for the RS-232 (COM1 device).

string_out

(str: txt);

Send a string to the aux device.

N_string_out

(str: txt; N: byte);

Send a string of length N to the aux device.

Functions:**clr_to_sen:**

boolean;

Check for transmit ready.

aux_ready:

boolean;

Test for aux port with a waiting character.

GET_aux:

char;

Get a character from the aux port.

string_in

(N: byte): txt;

Receive a string of length N from the aux device.

ACI Library Procedures:

The following is a list of library routines included in the TURBO PASCAL ACI library. All data types and global variables requiring definition are also included. A simple example routine is also included with the listings.

data_transfer (slave_device: byte;
MemType: byte;
MemOfs, MemSeg: integer;
AccessType: char;
bytes: integer);

This routine coordinates a data transfer. It is the basic call made for any transfer of data using the ACI. Depending on whether a read or write is requested, it calls one of the following routines: data_read_from_slave or data_out. Each of these two routines call the Init_comm, header and send_header routines to initiate the communication. All of these routines call upon the lower level communications procedures listed here.

data_read_from_slave (n_bytes: integer);
Coordinate the transfer of data from slave.

data_out (n_bytes: integer);
Coordinate the transfer of data to the slave.

Init_comm (slave_no: byte);
Initiate communications with a slave. Calls the ENQout procedure.

ENQout (slave_no: byte);
Send an enquiry string.

header (slave_no, MemType: byte;
MemOfs, MemSeg: integer;
RorW: char;
message_length: integer);

Generate the Header Block in preparation for transmission.

send_header (slave, Memtype: byte;
MemOfs, MemSeg: integer;
RW: char;
data_length: integer);

Send the Header Block. Calls the previous routine.

error_state (err: byte);
Set error status.

turn_on_interrupt:
Turn on the serial interrupt state.

turn_off_interrupt:
Turn off the serial interrupt state.

clean_aux

Clean the input buffer and support.

send_aux

(cha: char);

Send a character to the auxiliary port.

int_srv;

Service the serial on an interrupt basis.

init_8250

(rate: integer);

Set the baud rate for the RS-232 (COM1 device).

string_out

(str: txt);

Send a string to the aux device.

N_string_out

(str: txt; N: byte);

Send a string of length N to the aux device.

Functions:**clr_to_sen**

boolean;

Check for transmit ready.

aux_ready:

boolean;

Test for aux port with a waiting character.

GET_aux:

char;

Get a character from the aux port.

string_in

(N: byte): txt;

Receive a string of length N from the aux device.