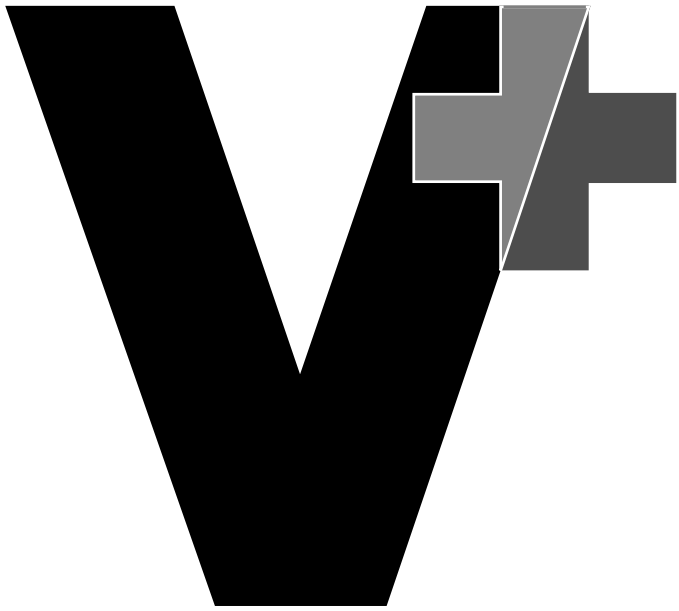
V+ Language

Reference Guide

Version 12.1



Part # 00962-01100, Rev A

September 1997

adept technology, inc.

150 Rose Orchard Way • San Jose, CA 95134 • USA • Phone (408) 432-0888 • Fax (408) 432-8707

Otto-Hahn-Strasse 23 • 44227 Dortmund • Germany • Phone (49) 231.75.89.40 • Fax(49) 231.75.89.450

41, rue du Saule Trapu • 91300 • Massy • France • Phone (33) 1.69.19.16.16 • Fax (33) 1.69.32.04.62

1-2, Aza Nakahara Mitsuva-Cho • Tovohashi, Aichi-Ken • 441-31 • Japan • (81) 532.65.2391 • Fax (81) 532.65.2390

Printed in the United States of America

Table of Contents

List of Figures

List of Tables

1 Introduction

Introduction

This reference guide is for use with V+ systems version 12.1 or later.

Related Publications

This reference guide is a companion to the ***V+ Language User's Guide***, which covers the principles of the V+ programming language and robot-control system.

In addition to being a complete programming language, V+ is also a complete operating system that controls equipment connected to Adept controllers. The ***V+ Operating System User's Guide*** and ***V+Operating System Reference Guide*** detail the V+ operating system. You must be familiar with the operating system in order to effectively use the V+ programming language.

The most current releases of some related publications may be for an earlier version of the V+ system. You need to use them in conjunction with the release notes published since those books were published.

You should have handy the manuals listed in **Table 1-1**.

Table 1-1. Related Publications

Manual	**Material Covered**
Adept MV Controller User's Guide	Instructions for setting up, configuring, and maintaining the controller V+ runs on.
AdeptForce VME User's Guide	Installation, operation, and programming of the AdeptForce VME product.
Adept Motion VME Developer's Guide	Installation, configuration, testing, and tuning of a motion device controlled by AdeptMotion VME.
AdeptNet User's Guide	Use and programming of the AdeptNet product.
AdeptVision Reference Guide (if your system is equipped with AdeptVision VXL)	Enhancements to the V+ language that are added when the AdeptVision option is installed.
Instructions for Adept Utility Programs	Instructions for running various setup and configuration software utilities
AdeptWindowsPC User's Guide	Instructions for using the AdeptWindowsPC product.
Release Notes for V+ Version 12.X	Late-breaking changes not in manuals; summary of changes.
Robot or motion device user's guides (if connected to your system)	Instructions for installing and maintaining the motion device connected to your system.

Conventions Used in This Manual

The following conventions are used throughout this manual to make this reference easier to use.

Notes, Cautions, and Warnings

Three levels of special notation are used in this manual. In descending order of importance, they are:

WARNING: If the actions indicated in a *warning* are not complied with, injury or major equipment damage could result. A warning typically describes the potential hazard, its possible effect, and the measures that must be taken to reduce the hazard.

CAUTION: If the action specified in a *caution* is not complied with, damage to your equipment or data could result.

NOTE: A note provides supplementary information, emphasizes or supplements a point or procedure, or gives a tip for easier operation.

Values, Variables, and Expressions

The parameters to V+ keywords can generally be satisfied with a specific value of the correct data type, a variable of the correct data type, or an expression that resolves to the correct type. Unless specifically stated, parameters can be replaced with a value, variable, or expression (of the correct type). The most common case where a parameter cannot be satisfied with all three options occurs when data is being returned in one of the parameters. In this case, a variable must be used; the parameter description states this restriction.

Integers and Real Values

In V+ integers and real values are not different data types. Real values satisfy parameters requiring integers by truncating the real value. Where real values are required, an integer is considered a special case of a real value with no fractional part.

Special Notation

Key combinations such as pressing the CTRL key and the Q key at the same time are shown as CTRL+Q.

Numbers shown in other than decimal format are preceded with a carat (^) and the letter H for hexadecimal, O for octal, or B for binary. For example, ^HF = ^B1111 = ^O17 = 15.

V+ Language Keyword Summary

Tables 1-2 to 1-9 summarize the keywords used by the V+ language. Appendix A is a quick reference to the V+ language. It lists all the keywords, their parameters, and a brief description of each keyword.

Keyword Groups

The V+ keyword summary groups keywords as follows:

Table 1-2 lists the keywords whose primary use is with tracking conveyor belt operations.

Table 1-3 lists the keywords used primarily with graphics-based systems to manage windows, graphics, and event-driven programming for a graphical user interface (GUI).

Table 1-4 lists the keywords whose primary use is for system input/output operations. These include disk I/O, serial I/O, digital I/O, and terminal I/O.

Table 1-5 lists the V+ logical operators.

Table 1-6 lists the keywords that control motion devices.

Table 1-7 lists the numerical operations (square root, modulo, trigonometric, and similar functions).

Table 1-8 lists the keywords primarily concerned with program control. Instructions for operations such as creating programs and program flow are in this group.

Table 1-9 lists the string operators (substring, string conversions, and similar functions).

Table 1-2. Conveyor Belt Operations

Keyword	Type	Function
BELT	S	Control the function of the conveyor tracking features of the V+ system.
BELT	RF	Return information about a conveyor belt being tracked with the conveyor tracking feature.
BELT.MODE	P	Set characteristics of the conveyor tracking feature of the V+ system.
BSTATUS	RF	Return information about the status of the conveyor tracking system.
DEFBELT	PI	Define a belt variable for use with a robot tracking conveyor belt.
SETBELT	PI	Set the encoder offset of the specified belt variable equal to the value of the expression.
WINDOW	PI	Set the boundaries of the operating region of the specified belt variable for conveyor tracking.
WINDOW	RF	Return a value that indicates where the location described by the belt-relative transformation value is relative to the predefined boundaries of the working range on a moving conveyor belt.
P: Parameter, PI: Program Instruction, RF: Real-Valued Function, S: Switch		

Table 1-3. Graphics Operations

Keyword	Type	Function
FDELETE	PI	Delete the specified disk file, the specified graphics window and all its child windows, or the specified graphics icon.
FOPEN	PI	Create and open a new graphics window, or open an existing graphics window, for subsequent input and/or output.
P: Parameter, PI: Program Instruction		

Table 1-3. Graphics Operations (Continued)

Keyword	Type	Function
FSET	PI	Set or modify attributes of a graphics window or serial line.
GARC	PI	Draw an arc or a circle in a graphics window.
GCHAIN	PI	Draw a chain of points in a graphics window to form a complex figure.
GCLEAR	PI	Clear an entire graphics window to the background color.
GCLIP	PI	Set the clipping rectangle for all graphics instructions (except GFLOOD) to suppress all subsequent graphics that fall outside the rectangle.
GCOLOR	PI	Set the foreground and background colors for subsequent graphics output.
GCOPY	PI	Copy one region of a window to another region in the same window.
GETEVENT	PI	Return information describing input from a graphics window.
GFLOOD	PI	Flood a region in a graphics window with color.
GGETLINE	PI	Return pixel information from a single pixel row in a graphics window.
GICON	PI	Draw a predefined graphic symbol (icon) in a graphics window.
GLINE	PI	Draw a single line segment in a graphics window.
GLINES	PI	Draw multiple line segments in a graphics window.
GLOGICAL	PI	Set the logical operation to be performed between new graphics output and graphics data already displayed, and select which bit planes are affected by graphics instructions.
GPANEL	PI	Draw a rectangular panel with shadowed or grooved edges.
GPOINT	PI	Draw a single point in a graphics window.

P: Parameter, PI: Program Instruction

Table 1-3. Graphics Operations (Continued)

Keyword	Type	Function
GRECTANGLE	PI	Draw a rectangle in a graphics window.
GSCAN	PI	Draw a number of horizontal lines in a graphics window to form a complex figure.
GSLIDE	PI	Draw a slide bar in preparation for receiving slide events.
GTEXTURE	PI	Set the opaque/transparent mode and the texture pattern for subsequent graphics output.
GTRANS	PI	Scale, rotate, and offset subsequent graphics instructions.
GTYPE	PI	Display a text string in a graphics window.
SCREEN.TIMEOUT	P	Establish the time-out period for blanking the screen of the graphics monitor.

P: Parameter, PI: Program Instruction

Table 1-4. System Input/Output Operations

Keyword	Type	Function
AIO.IN	RF	Read a channel from one of the analog IO boards.
AIO.INS	RF	Test whether an analog input or output channel is installed.
AIO.OUT	PI	Write to a channel on one of the analog IO boards.
ATTACH	PI	Make a device available for use by the application program.
BITS	PI	Set or clear a group of digital signals based on a value.
BITS	RF	Read multiple digital signals and return the value corresponding to the binary bit pattern present on the signals.
$DEFAULT	SF	Return a string containing the current system default device, unit, and directory path for disk file access.

P: Parameter, PI: Program Instruction, RF: Real-Valued Function, SF: String Function

Table 1-4. System Input/Output Operations (Continued)

Keyword	Type	Function
DEF.DIO	PI	Assign third-party digital I/O boards to standard V+ signal numbers, for use by standard V+ instructions, functions, and monitor commands.
DETACH	PI	Release a specified device from the control of the application program.
DEVICE	PI	Send a command or data to an external device and, optionally, return data back to the program. (The actual operation performed depends on the device referenced.)
DEVICE	RF	Return a real value from a specified device. The value may be data or status information, depending upon the device and the parameters.
DEVICES	PI	Send commands or data to an external device and optionally return data. The actual operation performed depends on the device referenced.
FCLOSE	PI	Close the disk file, graphics window, or graphics icon currently open on the specified logical unit.
FCMND	PI	Generate a device-specific command to the input/output device specified by the logical unit.
FEMPTY	PI	Empty any internal buffers in use for a disk file or a graphics window by writing the buffers to the file or window if necessary.
FOPENA	PI	Open a disk file for read-only, read-write, read-write-append, or read-directory, respectively, as indicated by the last letter of the instruction name.
FOPEND	PI	Open a disk file for read-only, read-write, read-write-append, or read-directory, respectively, as indicated by the last letter of the instruction name.
FOPENR	PI	Open a disk file for read-only, read-write, read-write-append, or read-directory, respectively, as indicated by the last letter of the instruction name.

P: Parameter, PI: Program Instruction, RF: Real-Valued Function, SF: String Function

Table 1-4. System Input/Output Operations (Continued)

Keyword	Type	Function
FOPENW	PI	Open a disk file for read-only, read-write, read-write-append, or read-directory, respectively, as indicated by the last letter of the instruction name.
FSEEK	PI	Position a file open for random access and initiate a read operation on the specified record.
FSET	PI	Set serial I/O port configuration parameters.
GETC	RF	Return the next character (byte) from a device or input record on the specified logical unit.
IGNORE	PI	Cancel the effect of a REACT or REACTI instruction.
IOGET_	RF	Return a value from a device on the VME bus.
$IOGETS	SF	Return a string value from a device on the VME bus.
IOPUT_	PI	Send a value to a device on the VME bus.
IOSTAT	RF	Return status information for the last input/output operation for a device associated with a logical unit.
IOTAS	RF	Control access to shared devices on the VME bus.
KERMIT.RETRY	P	Establish the maximum number of times the (local) Kermit driver should retry an operation before reporting an error.
KERMIT.TIMEOUT	P	Establish the delay parameter that the V+ driver for the Kermit protocol will send to the remote server.
KEYMODE	PI	Set the behavior of a group of keys on the manual control pendant.
NETWORK	RF	Return network status and IP address information.
PENDANT	RF	Return input from the manual control pendant.
PROMPT	PI	Display a string on the system terminal and wait for operator input.
READ	PI	Read a record from an open file or from an attached device that is not file oriented.

P: Parameter, PI: Program Instruction, RF: Real-Valued Function, SF: String Function

Table 1-4. System Input/Output Operations (Continued)

Keyword	Type	Function
RESET	PI	Turn off all the external output signals.
SETDEVICE	PI	Initialize a device or set device parameters. (The actual operation performed depends on the device referenced.)
SIG	RF	Return the logical AND of the states of the indicated digital signals.
SIG.INS	RF	Return an indication of whether or not a digital I/O signal is configured for use by the system, or whether or not a software signal is available in the system.
SIGNAL	PI	Turn on or off external digital output signals or internal software signals.
TYPE	PI	Display the information described by the output specifications on the system terminal. A blank line is output if no argument is provided.
WRITE	PI	Write a record to an open file or to an attached device that is file oriented.
P: Parameter, PI: Program Instruction, RF: Real-Valued Function, SF: String Function		

Table 1-5. Logical Operations

Keyword	Type	Function
AND	O	Perform the logical AND operation on two values.
BAND	O	Perform the binary AND operation on two values.
BMASK	RF	Create a bit mask by specifying which bits to set.
BOR	O	Perform the binary OR operation on two values.
BXOR	O	Perform the binary exclusive-OR operation on two values.
COM	O	Perform the binary complement operation on a value.
FALSE	RF	Return the value used by V+ to represent a logical false result.
O: Operator, RF: Real-Valued Function		

Table 1-5. Logical Operations (Continued)

Keyword	Type	Function
MOD	O	Compute the modulus of two values.
NOT	O	Perform logical negation of a value.
OFF	RF	Return the value used by V+ to represent a logical false result.
ON	RF	Return the value used by V+ to represent a logical true result.
OR	O	Perform the logical OR operation on two values.
TRUE	RF	Return the value used by V+ to represent a logical true result.
XOR	O	Perform the logical exclusive-OR operation on two values.
O: Operator, RF: Real-Valued Function		

Table 1-6. Motion Control Operations

Keyword	Type	Function
ABOVE	PI	Request a change in the robot configuration during the next motion so that the elbow is above the line from the shoulder to the wrist.
ACCEL	PI	Set acceleration and deceleration for robot motions.
ACCEL	RF	Return the current robot acceleration or deceleration setting.
ALIGN	PI	Align the robot tool Z axis with the nearest world axis.
ALTER	PI	Specify the magnitude of the real-time path modification that is to be applied to the robot path during the next trajectory computation.
ALTOFF	PI	Terminate real-time path-modification mode (alter mode).
ALTON	PI	Enable real-time path-modification mode (alter mode), and specify the way in which ALTER coordinate information will be interpreted.
CF: Conversion Factor, P: Parameter, PF: Precision-Point Function, PI: Program Instruction, RF: Real-Valued Function, S: Switch, TF: Transformation Function		

Table 1-6. Motion Control Operations (Continued)

Keyword	Type	Function
APPRO	PI	Start a robot motion toward a location defined relative to specified location.
APPROS	PI	Start a robot motion toward a location defined relative to specified location.
BASE	PI	Translate and rotate the World reference frame relative to the robot.
BASE	TF	Return the transformation value that represents the translation and rotation set by the last BASE command or instruction.
BELOW	PI	Request a change in the robot configuration during the next motion so that the elbow is below the line from the shoulder to the wrist.
BRAKE	PI	Abort the current robot motion.
BREAK	PI	Suspend program execution until the current motion completes.
CALIBRATE	PI	Initialize the robot positioning system.
CLOSE	PI	Close the robot gripper.
CLOSEI	PI	Close the robot gripper.
COARSE	PI	Enable a low-precision feature of the robot hardware servo.
CONFIG	RF	Return a value that provides information about the robot's geometric configuration or the status of the motion servo-control features.
CP	S	Control the continuous-path feature.
CPOFF	PI	Instruct the V+ system to stop the robot at the completion of the next motion instruction (or all subsequent motion instructions) and null position errors.

CF: Conversion Factor, P: Parameter, PF: Precision-Point Function, PI: Program Instruction, RF: Real-Valued Function, S: Switch, TF: Transformation Function

Table 1-6. Motion Control Operations (Continued)

Keyword	Type	Function
CPON	PI	Instruct the V+ system to execute the next motion instruction (or all subsequent motion instructions) as part of a continuous path.
DECEL.100	S	Enable or disable the use of 100 percent as the maximum deceleration for the ACCEL program instruction.
DECOMPOSE	PI	Extract the (real) values of individual components of a location value.
DELAY	PI	Cause robot motion to stop for the specified period of time.
DEPART	PI	Start a robot motion away from the current location.
DEPARTS	PI	Start a robot motion away from the current location.
DEST	TF	Return a transformation value representing the planned destination location for the current robot motion.
DISTANCE	RF	Determine the distance between the points defined by two location values.
DRIVE	PI	Move an individual joint of the robot.
DRY.RUN	S	Control whether or not V+ communicates with the robot.
DURATION	PI	Set the minimum execution time for subsequent robot motions.
DURATION	RF	Return the current setting of one of the motion DURATION specifications.
DX	RF	Return a displacement component of a given transformation value.
DY	RF	Return a displacement component of a given transformation value.

CF: Conversion Factor, P: Parameter, PF: Precision-Point Function, PI: Program Instruction, RF: Real-Valued Function, S: Switch, TF: Transformation Function

Table 1-6. Motion Control Operations (Continued)

Keyword	Type	Function
DZ	RF	Return a displacement component of a given transformation value.
ESTOP	PI	Assert the emergency-stop signal to stop the robot.
FINE	PI	Enable a high-precision feature of the robot hardware servo.
FLIP	PI	Request a change in the robot configuration during the next motion so that the pitch angle of the robot wrist has a negative value.
FRAME	TF	Return a transformation value defined by four positions.
GAIN.SET	PI	Select a set of gain parameters for one or more joints of the current robot.
HAND	RF	Return the current hand opening.
HAND.TIME	P	Establish the duration of the motion delay that occurs during OPENI, CLOSEI, and RELAXI instructions.
HERE	PI	Set the value of a transformation or precision-point variable equal to the current robot location.
HERE	TF	Return a transformation value that represents the current location of the robot tool point.
HOUR.METER	RF	Return the current value of the robot hour meter.
IDENTICAL	RF	Determine if two location values are exactly the same.
INRANGE	RF	Return a value that indicates if a location can be reached by the robot, and if not, why not.
INVERSE	TF	Return the transformation value that is the mathematical inverse of the given transformation value.
IPS	CF	Specify the units for a SPEED instruction as inches per second.

CF: Conversion Factor, P: Parameter, PF: Precision-Point Function, PI: Program Instruction, RF: Real-Valued Function, S: Switch, TF: Transformation Function

Table 1-6. Motion Control Operations (Continued)

Keyword	Type	Function
LATCH	TF	Return a transformation value representing the location of the robot at the occurrence of the last external trigger.
LATCHED	RF	Return the status of the external trigger and of the information it causes to be latched.
LEFTY	PI	Request a change in the robot configuration during the next motion so that the first two links of a SCARA robot resemble a human's left arm.
MMPS	CF	Specify the units for a SPEED instruction as millimeters per second.
MOVE	PI	Initiate a robot motion to the position and orientation described by the given location.
MOVES	PI	Initiate a robot motion to the position and orientation described by the given location.
MOVEF	PI	Initiate a three-segment pick-and-place robot motion to the specified destination, moving the robot at the fastest allowable speed.
MOVESF	PI	Initiate a three-segment pick-and-place robot motion to the specified destination, moving the robot at the fastest allowable speed.
MOVET	PI	Initiate a robot motion to the position and orientation described by the given location and simultaneously operate the hand.
MOVEST	PI	Initiate a robot motion to the position and orientation described by the given location and simultaneously operate the hand.
MULTIPLE	PI	Allow full rotations of the robot wrist joints.
NOFLIP	PI	Request a change in the robot configuration during the next motion so that the pitch angle of the robot wrist has a positive value.

CF: Conversion Factor, P: Parameter, PF: Precision-Point Function, PI: Program Instruction, RF: Real-Valued Function, S: Switch, TF: Transformation Function

Table 1-6. Motion Control Operations (Continued)

Keyword	Type	Function
NONULL	PI	Instruct the V+ system not to wait for position errors to be nulled at the end of continuous-path motions.
NOOVERLAP	PI	Generate a program error if a motion is planned that will cause selected multiturn axes to turn more than ± 180 degrees (the long way around) in order to avoid a limit stop.
NORMAL	TF	Correct a transformation for any mathematical round-off errors.
NOT.CALIBRATED	P	Indicate (or assert) the calibration status of the robots connected to the system.
NULL	TF	Return a null transformation value—one with all zero components.
OPEN	PI	Open the robot gripper.
OPENI	PI	Open the robot gripper.
OVERLAP	PI	Disable the NOOVERLAP limit-error checking either for the next motion or for all subsequent motions.
PAYLOAD	PI	Set an indication of the current robot payload.
#PDEST	PF	Return a precision-point value representing the planned destination location for the current robot motion.
#PLATCH	PF	Return a precision-point value representing the location of the robot at the occurrence of the last external trigger.
POWER	S	Control or monitor the status of Robot Power.
#PPOINT	PF	Return a precision-point value composed from the given components.
REACTI	PI	Initiate continuous monitoring of a specified digital signal. Automatically stop the current robot motion if the signal transitions properly and, optionally, trigger a subroutine call.

CF: Conversion Factor, P: Parameter, PF: Precision-Point Function, PI: Program Instruction, RF: Real-Valued Function, S: Switch, TF: Transformation Function

Table 1-6. Motion Control Operations (Continued)

Keyword	Type	Function
READY	PI	Move the robot to the READY location above the workspace, which forces the robot into a standard configuration.
RELAX	PI	Limp the pneumatic hand.
RELAXI	PI	Limp the pneumatic hand.
RIGHTY	PI	Request a change in the robot configuration during the next motion so that the first two links of the robot resemble a human's right arm.
ROBOT	S	Enable or disable one robot or all robots.
RX	TF	Return a transformation describing a rotation.
RY	TF	Return a transformation describing a rotation.
RZ	TF	Return a transformation describing a rotation.
SCALE	TF	Return a transformation value equal to the transformation parameter with the position scaled by the scale factor.
SCALE.ACCEL	S	Scale acceleration and deceleration as a function of program speed when program speed is below 100%.
SCALE.ACCEL.ROT	S	Specify whether or not the SCALE.ACCEL switch takes into account the Cartesian rotational speed during straight-line motions.
SELECT	PI	Select the unit of the named device for access by the current task.
SET	PI	Set the value of the location variable on the left equal to the location value on the right of the equal sign.
#SET.POINT	PF	Return the commanded joint-angle positions computed by the trajectory generator during the last trajectory-evaluation cycle.

CF: Conversion Factor, P: Parameter, PF: Precision-Point Function, PI: Program Instruction, RF: Real-Valued Function, S: Switch, TF: Transformation Function

Table 1-6. Motion Control Operations (Continued)

Keyword	Type	Function
SET.SPEED	S	Control whether or not the monitor speed can be changed from the manual control pendant. The monitor speed cannot be changed when the switch is disabled.
SHIFT	TF	Return a transformation value resulting from shifting the position of the transformation parameter by the given shift amounts.
SINGLE	PI	Limit rotations of the robot wrist joint to the range –180 degrees to +180 degrees.
SOLVE.ANGLES	PI	Compute the robot joint positions (for the current robot) that are equivalent to a specified transformation.
SOLVE.FLAGS	RF	Return bit flags representing the robot configuration specified by an array of joint positions.
SOLVE.TRANS	PI	Compute the transformation equivalent to a given set of joint positions for the current robot.
SPEED	PI	Set the nominal speed for subsequent robot motions.
SPEED	RF	Return one of the system motion speed factors.
SPIN	PI	Rotate one or more continuous-turn joints of the selected robot at a specified speed.
STATE	RF	Return a value that provides information about the robot system state.
TOOL	PI	Set the internal transformation used to represent the location and orientation of the tool tip relative to the tool-mounting flange of the robot.
TOOL	TF	Return the value of the transformation specified in the last TOOL command or instruction.
TRANS	TF	Return a transformation value computed from the given X, Y, Z position displacements and y, p, r orientation rotations.

CF: Conversion Factor, P: Parameter, PF: Precision-Point Function, PI: Program Instruction, RF: Real-Valued Function, S: Switch, TF: Transformation Function

Table 1-6. Motion Control Operations (Continued)

Keyword	Type	Function
TRANSB	TF	Return a transformation value represented by a 48-byte string.
UNIDIRECT	PI	Specify that a continuous-turn joint is turning only in a single direction.

CF: Conversion Factor, P: Parameter, PF: Precision-Point Function, PI: Program Instruction, RF: Real-Valued Function, S: Switch, TF: Transformation Function

Table 1-7. Numeric Value Operations

Keyword	Type	Function
ABS	RF	Return absolute value.
ATAN2	RF	Return the size of the angle (in degrees) that has its trigonometric tangent equal to value_1/value_2.
BCD	RF	Convert a real value to Binary Coded Decimal (BCD) format.
COS	RF	Return the trigonometric cosine of a given angle.
DBLB	RF	Return the value of eight bytes of a string interpreted as an IEEE double-precision floating-point number.
$DBLB	SF	Return an 8-byte string containing the binary representation of a real value in double-precision IEEE floating-point format.
DCB	RF	Convert BCD digits into an equivalent integer value.
FLTB	RF	Return the value of four bytes of a string interpreted as an IEEE single-precision floating-point number.
$FLTB	SF	Return a 4-byte string containing the binary representation of a real value in single-precision IEEE floating-point format.
FRACT	RF	Return the fractional part of the argument.
INT	RF	Return the integer part of the value.
INTB	RF	Return the value of two bytes of a string interpreted as a signed 16-bit binary integer.

RF: Real-Valued Function, SF: String Function

Table 1-7. Numeric Value Operations (Continued)

Keyword	Type	Function
LNGB	RF	Return the value of four bytes of a string interpreted as a signed 32-bit binary integer.
$LNGB	SF	Return a 4-byte string containing binary representation of a 32-bit integer.
MAX	RF	Return the maximum value contained in the list of values.
MIN	RF	Return the minimum value contained in the list of values.
OUTSIDE	RF	Test a value to see if it is outside a specified range.
PI	RF	Return the value of the mathematical constant pi (3.141593).
RANDOM	RF	Return a pseudo random number.
SIGN	RF	Return the value 1, with the sign of the value parameter.
SIN	RF	Return the trigonometric sine of a given angle.
SQR	RF	Return the square of the parameter.
SQRT	RF	Return the square root of the parameter.
VAL	RF	Return the real value represented by the characters in the input string.
RF: Real-Valued Function, SF: String Function		

Table 1-8. Program Control Operations

Keyword	Type	Function
ABORT	PI	Terminate execution of a program task.
ALWAYS	K	Used with certain program instructions to specify a long-term effect.
ANY	PI	Signal the beginning of an alternative group of instructions for the CASE structure.
K: Keyword, P: Parameter, PI: Program Instruction, RF: Real-Valued Function, S: Switch, SF: String Function		

Table 1-8. Program Control Operations (Continued)

Keyword	Type	Function
AUTO	PI	Declare temporary variables that are automatically created on the program stack when the program is entered.
AUTO.POWER.OFF	S	Control whether or not V+ disables high power when certain motion errors occur.
BY	K	Complete the syntax of the SCALE and SHIFT functions.
CALL	PI	Suspend execution of the current program and continue execution with a new program (that is, a subroutine).
CALLP	PI	Call a program given a pointer to the program.
CALLS	PI	Suspend execution of the current program and continue execution with a new program (that is, a subroutine) specified with a string value.
CASE	PI	Initiate processing of a CASE structure by defining the value of interest.
CLEAR.EVENT	PI	Clear an event associated with the specified task.
CYCLE.END	PI	Terminate the specified control program the next time it executes a STOP program instruction (or its equivalent). Suspend processing of an application program or command program until a program completes execution.
DEFINED	RF	Determine whether a variable has been defined.
DISABLE	PI	Turn off one or more system control switches.
DO	PI	Introduce a DO program structure.
DOS	PI	Execute a program instruction defined by a string expression.
ELSE	PI	Separate the alternate group of statements in an IF ... THEN control structure.
ENABLE	PI	Turn on one or more system control switches.
END	PI	Mark the end of a control structure.

K: Keyword, P: Parameter, PI: Program Instruction, RF: Real-Valued Function, S: Switch, SF: String Function

Table 1-8. Program Control Operations (Continued)

Keyword	Type	Function
.END	PI	Mark the end of a V+ program.
ERROR	RF	Return the error number of a recent error that caused program execution to stop or caused a REACTE reaction.
$ERROR	SF	Return the error message associated with the given error code.
EXECUTE	PI	Begin execution of a control program.
EXIT	PI	Branch to the statement following the nth nested loop of a control structure.
FOR	PI	Execute a group of program instructions a certain number of times.
FREE	RF	Return the amount of unused free memory storage space.
GET.EVENT	RF	Return events that are set for the specified task.
GLOBAL	PI	Declare a variable to be global and specify the type of the variable.
GOTO	PI	Perform an unconditional branch to the program step identified by the given label.
HALT	PI	Stop program execution and do not allow the program to be resumed.
ID	RF	Return values that identify the configuration of the current system.
$ID	SF	
IF GOTO	PI	Branch to the specified label if the value of a logical expression is TRUE (nonzero).
IF... THEN	PI	Conditionally execute a group of instructions (or one of two groups) depending on the result of a logical expression.
INSTALL	PI	Install an Adept software option.

K: Keyword, P: Parameter, PI: Program Instruction, RF: Real-Valued Function, S: Switch, SF: String Function

Table 1-8. Program Control Operations (Continued)

Keyword	Type	Function
INTERACTIVE	S	Control the display of message headers on the system terminal and requests for confirmation before performing certain operations.
INT.EVENT	PI	Send a SET.EVENT instruction to the current task if an interrupt occurs on a specified VME bus vector or a specified digital I/O transitions to positive.
KILL	PI	Clear a program execution stack and detach any I/O devices that are attached.
LAST	RF	Return the highest index used for an array (dimension).
LOCAL	PI	Declare permanent variables that are defined only within the current program.
LOCK	PI	Set the program reaction lock-out priority to the value given.
MC	PI	Introduce a monitor command within a command program.
MCP.MESSAGE	S	Control how system error messages are handled when the controller keyswitch is not in the PENDANT position.
MCS	PI	Invoke a monitor command from an application program.
MCS.MESSAGE	S	Enable or disable output to the system terminal from monitor commands executed with the MCS instruction.
MESSAGES	S	Enable or disable output to the system terminal from TYPE instructions.
MONITORS	S	Enable or disable selecting of multiple monitor windows.
NEXT	PI	Branch to the top of the next iteration of a looping control structure.
NULL	PI	Instruct the V+ system to wait for position errors to be nulled at the end of continuous path motions.

K: Keyword, P: Parameter, PI: Program Instruction, RF: Real-Valued Function, S: Switch, SF: String Function

Table 1-8. Program Control Operations (Continued)

Keyword	Type	Function
PARAMETER	PI	Set the value of a system parameter.
PARAMETER	RF	Return the current setting of the named system parameter.
PAUSE	PI	Stop program execution but allow the program to be resumed.
PRIORITY	RF	Return the current reaction lock-out priority for the program.
REACT	PI	Initiate continuous monitoring of a specified digital signal and automatically trigger a subroutine call if the signal properly transitions.
REACTE	PI	Initiate the monitoring of errors that occur during execution of the current program task.
RELEASE	PI	Allow the next available program task to run.
RETRY	S	Control whether the PROGRAM START button causes a program to resume.
RETURN	PI	Terminate execution of the current subroutine and resume execution of the last-suspended program at the step following the CALL or CALLS instruction that caused the subroutine to be invoked.
RETURNE	PI	Terminate execution of an error reaction subroutine and resume execution of the last-suspended program at the step following the instruction that caused the subroutine to be invoked.
RUNSIG	PI	Turn on (or off) the specified digital signal as long as execution of the invoking program task continues.
SEE	PI	Invoke the screen-oriented program editor to allow a program to be created, viewed, or modified.
SELECT	RF	Return the unit number that is currently selected by the current task for the device named.
SET.EVENT	PI	Set an event associated with the specified task.

K: Keyword, P: Parameter, PI: Program Instruction, RF: Real-Valued Function, S: Switch, SF: String Function

Table 1-8. Program Control Operations (Continued)

Keyword	Type	Function
STATUS	RF	Return status information for an application program.
STOP	PI	Terminate execution of the current program cycle.
SWITCH	PI	Enable or disable a system switch based on a value.
SWITCH	RF	Return an indication of the setting of a system switch.
TAS	RF	Return the current value of a real-valued variable and assign it a new value. The two actions are done indivisibly so no other program task can modify the variable at the same time.
TASK	RF	Return information about a program execution task.
TERMINAL	P	Determine how V+ will interact with the system terminal.
TIME	PI	Set the date and time.
TIME	RF	Return an integer value representing either the date or the time specified in the given string parameter.
$TIME	SF	Return a string value containing either the current system date and time or the specified date and time.
TIMER	PI	Set the specified system timer to the given time value.
TIMER	RF	Return the current time value (in seconds) of the specified system timer.
TPS	RF	Return the number of ticks of the system clock that occur per second (Ticks Per Second).
TRACE	S	Control the display of program steps on the system terminal during program execution.
UNTIL	PI	Indicate the end of a DO ... UNTIL control structure and specify the expression that is evaluated to determine when to exit the loop. The loop continues to be executed until the expression value is non-zero.
UPPER	S	Control whether or not string comparisons are performed ignoring the case of each character.

K: Keyword, P: Parameter, PI: Program Instruction, RF: Real-Valued Function, S: Switch, SF: String Function

Table 1-8. Program Control Operations (Continued)

Keyword	Type	Function
VALUE	PI	Indicate the values that a CASE statement expression must match in order for the program statements immediately following to be executed.
WAIT	PI	Put the program into a wait loop until the condition is TRUE.
WAIT.EVENT	PI	Suspend program execution until a specified event has occurred or until a specified amount of time has elapsed.
WHILE	PI	Initiate processing of a WHILE structure if the condition is TRUE or skipping of the WHILE structure if the condition is initially FALSE.

K: Keyword, P: Parameter, PI: Program Instruction, RF: Real-Valued Function, S: Switch, SF: String Function

Table 1-9. String Operations

Keyword	Type	Function
ASC	RF	Return an ASCII character value from within a string.
$CHR	SF	Return a one-character string having a given value.
$DBLB	SF	Return an 8-byte string containing the binary representation of a real value in double-precision IEEE floating-point format.
$DECODE	SF	Extract part of a string as delimited by given break characters.
$ENCODE	SF	Return a string created from output specifications. The string produced is similar to the output of a TYPE instruction.
$FLTB	SF	Return a 4-byte string containing the binary representation of a real value in single-precision IEEE floating-point format.
$INTB	SF	Return a 2-byte string containing the binary representation of a 16-bit integer.

PI: Program Instruction, RF: Real-Valued Function, SF: String Function

Table 1-9. String Operations (Continued)

Keyword	Type	Function
LEN	RF	Return the number of characters in the given string.
$LNGB	SF	Return a 4-byte string containing the binary representation of a 32-bit integer.
$MID	SF	Return a substring of the specified string.
PACK	PI	Replace a substring within an array of (128-character) string variables or within a (nonarray) string variable.
POS	RF	Return the starting character position of a substring in a string.
STRDIF	RF	Compare two strings byte by byte for the purpose of sorting.
$SYMBOL	SF	Determine the user symbol that is referenced by a pointer previously obtained with the SYMBOL.PTR real-valued function.
SYMBOL.PTR	SF	Determine the value of a pointer to a user symbol in V+ memory.
$TRANSB	SF	Return a 48-byte string containing the binary representation of a transformation value.
$TRUNCATE	SF	Return all characters in the input string until an ASCII NUL (or the end of the string) is encountered.
$UNPACK	SF	Return a substring from an array of 128-character string variables.
PI: Program Instruction, RF: Real-Valued Function, SF: String Function		

How Can I Get Help?

The following section tells you who to call if you need help.

Within the Continental United States

Adept Technology maintains a Customer Service Center at its headquarters in San Jose, California. The phone numbers are:

Service Calls

(800) 232-3378 (24 hours a day, 7 days a week)
(408) 433-9462 FAX

> **NOTE:** When calling with a controller-related question, please have the serial number of the controller. If your system includes an Adept robot, also have the serial number of the robot. The serial numbers can be determined by using the ID command (see the *V+ Operating System User's Guide*).

Application Questions

If you have an application question, you can contact the Adept Applications Engineering Support Center for your region:

Adept Office	Phone #, Hours	Region
San Jose, CA	Voice (408) 434-5033 Fax (408) 434-6248 8:00 A.M. – 5:00 P.M. PST	Western Region States: AR, AZ, CA, CO, ID, KS, LA, MO, MT, NE, NM, NV, OK, OR, TX, UT, WA, WY
Cincinnati, OH	Voice (513) 792-0266 Fax (513) 792-0274 8:00 A.M. – 5:00 P.M. EST	Midwestern Region States: AL, IA, IL, IN, KY, MI, MN, MS, ND, West NY, OH, West PA, SD, TN, WI
Southbury, CT	Voice (203) 264-0564 Fax (203) 264-5114 8:00 A.M. – 5:00 P.M. EST	Eastern Region States: CT, DE, FL, GA, MD, ME, NC, NH, MA, NJ, East NY, East PA, RI, SC, VA, VT, WV

Applications Internet E-Mail Address

If you have access to the Internet, you can send application questions by e-mail to:

```
adeptinfo@infolab.com
```

This method also enables you to attach a file, such as a portion of V+ program code, to your message.

NOTE: Please attach only information that is formatted as text.

Training Information

For information regarding Adept Training Courses in the USA, please call (408) 474-3246 or fax Adept at (408) 474-3226.

Within Europe

Adept Technology maintains a Customer Service Center in Dortmund, Germany. The phone numbers are:

(49) 231 / 75 89 40 from within Europe (Monday to Friday, 8:00 A.M. to 5:00 P.M.)
(49) 231/75 89 450 FAX

France

For customers in France, Adept Technology maintains a Customer Service Center in Massy, France. The phone numbers are:

(33) 1 69 19 16 16 (Monday to Friday, 8:30 A.M. to 5:30 P.M., CET)
(33) 1 69 32 04 62 FAX

Outside Continental United States or Europe

For service calls, application questions, and training information, call the Adept Customer Service Center in San Jose, California USA:

1 (408) 434-5000
1 (408) 433-9462 FAX (service requests)
1 (408) 434-6248 FAX (application questions)

Adept Fax on Demand

Adept maintains a fax-back information system for customer use. The phone numbers are (800) 474-8889 (toll free) and (503) 207-4023 (toll call). Application utility programs, product technical information, customer service information, and corporate information is available through this automated system. There is no charge for this service (except for any long-distance toll charges). Simply call either number and follow the instructions to have information faxed directly to you.

Adept on Demand Web Page

If you have access to the Internet, you can view Adept's web page at the following address:

```
http://www.adept.com
```

The web site contains sales, customer service, and technical support information.

2 Descriptions of V+ Keywords

Introduction

This chapter details the keywords in the V+ programming language. The functional groups of programming keywords are:

- Program Instructions
- Functions
- System Parameters
- System Switches

This manual often refers to *monitor commands*. Monitor commands are part of the V+ operating system. The V+ operating system commands are detailed in the ***V+ Operating System Reference Guide***.

If your system is equipped with AdeptVision VXL, additional program instructions, functions, switches, parameters, and monitor commands are detailed in the ***AdeptVision Reference Guide***.

The keywords are presented in alphabetical order, with the description for each keyword starting on a new page.

Each keyword is described, starting with the syntax, as shown on pages **43** and **44**.

Syntax

This section presents the syntax of the keyword. The keyword is shown in uppercase, and the arguments are shown in lowercase. The keyword must be entered exactly as shown.[1] Parentheses must be placed exactly as shown. Required keywords, parameters, and marks such as equal signs and parentheses are shown in bold type; optional keywords, parameters, and marks are shown in regular type. In the example:

`KEYWORD req.param1 = req.param2` `OPT.KEYWORD opt.param`

`KEYWORD`	must be entered exactly as shown,[1]
`req.param1`	must be replaced with a value, variable, or expression,
the equal sign	must be entered,
`req.param2`	must be replaced with a value, variable, or expression,
`OPT.KEYWORD`	can be omitted but must be entered exactly as shown if used,
`opt.param`	may be replaced with a value, variable, or expression but assumes a default value if not used.

The keyword type (function, program instruction, and so on) is shown at the top of the page.

An abbreviated syntax is shown for some keywords. This is done when the abbreviated form is the most commonly used variation of the complete syntax.

Function

This section gives a brief description of the keyword.

Usage Considerations

This section lists any restrictions on the keyword's use. If specific hardware or other options are required, they are listed here.

[1] In the SEE editor, instructions can be abbreviated to a length that uniquely identifies the keyword. The SEE editor automatically expands the instruction to its full length.

Parameters

The requirements for input and output parameters are explained in this section. If a parameter is optional, it is noted here. When an instruction line is entered, optional parameters do not have to be specified and the system will assume a default. Unspecified parameters at the end of an argument list can be ignored. Unspecified parameters in the middle of an argument list must be represented by commas. For example, the following keyword has four parameters—the first and third are used, and the second and fourth are left unspecified:

```
SAMPLE.INST var_1,,"test"
```

String and numeric input parameters can be constant values (3.32, part_1, etc.) or any legitimate variable names. The data type of the constant or variable must agree with that expected by the instruction. String variables must be preceded by with a $. Precision-point variables must be preceded with a #. Belt variables must be preceded with a %. String constants must be enclosed in quotes. Real and integer constants can be used without modification. (Some V+ keywords cannot be used as variable names—see **"V+ Language Keyword Summary" on page 14** for a complete list of keywords.)

Details

This section describes the function of the keyword in detail.

Examples

Examples of correctly formed instruction lines are presented in this section.

Related Keywords

Additional keywords that are similar or are frequently used in conjunction with this instruction are listed here.

Any related keywords that are monitor commands are described in the ***V+ Operating System Reference Guide***.

Syntax

ABORT task_num

Function

Terminate execution of an executing program task.

Usage Considerations

ABORT is ignored if no program is executing as the specified task.

ABORT does *not* force DETACH or FCLOSE operations on the disk or serial communication logical units. If the program has one or more files open and you decide not to resume execution of the program, use a KILL command to close all the files and detach the logical units.

Parameter

task_num	Optional real value, variable, or expression (interpreted as an integer) that specifies which program task is to be terminated. The default task is 0.

Details

Terminates execution of the specified active executable program after completion of the step currently being executed. If the task is controlling a robot, robot motion terminates at the completion of the current motion. (Program execution can be resumed with the PROCEED command.)

Related Keywords

ABORT	(monitor command, see the *V+ Operating System Reference Guide*)
CYCLE.END	(monitor command and program instruction)
EXECUTE	(monitor command and program instruction)
KILL	(monitor command and program instruction)
PANIC	(monitor command, see the *V+ Operating System Reference Guide*)
PROCEED	(monitor command, see the *V+ Operating System Reference Guide*)
STATUS	(monitor command and real-valued function)

Syntax

ABOVE

Function

Request a change in the robot configuration during the next motion so that the elbow is above the line from the shoulder to the wrist.

Usage Considerations

Configuration changes cannot be made during straight-line motions.

If the selected robot does not support an ABOVE configuration, this instruction is ignored by the robot (SCARA robots, for example, cannot have an ABOVE configuration).

The ABOVE instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the ABOVE instruction causes an error.

Figure 2-1 shows the ABOVE and BELOW configurations.

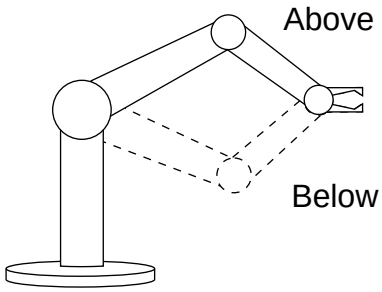


Figure 2-1. ABOVE/BELOW

Related Keywords

BELOW (program instruction)

CONFIG (real-valued function)

SELECT (program instruction and real-valued function)

Syntax

ABS (value)

Function

Return absolute value.

Parameter

value Real-valued expression.

Details

Returns the absolute value (magnitude) of the argument provided.

Examples

```
ABS(0.123)     ;Returns 0.123
ABS(-5.462)    ;Returns 5.462
ABS(1.3125E-2) ;Returns 0.013125
belt.length = part.size/ABS(belt.scale)
```

Syntax

```
ACCEL (profile) acceleration, deceleration
```

Function

Set acceleration and deceleration for robot motions. Optionally, specify a defined acceleration profile.

Usage Considerations

The ACCEL instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the ACCEL instruction causes the error *Robot not attached to this program*.

Before an acceleration/deceleration profile can be used, it must be defined for the selected robot (profile 0 is always defined). The SPEC utility program can be used by the robot manufacturer to define new or alter existing profiles (see the *Instructions for Adept Utility Programs*).

Parameters

`profile` Optional integer specifying the acceleration profile to use. Acceptable values are 0 to 8 (depending on the number of defined profiles). The default is the last specified profile (see Details for the number of the start-up profile). If a profile is specified that has not been defined, profile 0 is used.

`acceleration` Optional real value, variable, or expression considered a percentage of the maximum possible acceleration.

`deceleration` Optional real value, variable, or expression considered a percentage of the maximum possible deceleration.

The value should normally be in the range of 1 to 100 (upper limits greater than 100 may be established by the robot manufacturer). If an out-of-range value is specified, the nearest extreme value will be used.

If a parameter is omitted, its current setting remains in effect.

Details

If profile 0 is used, a square wave acceleration profile will be generated at the beginning and end of the motion.

If a profile is specified, that profile is invoked for subsequent robot motions. Defined profiles set the maximum rate of change of the acceleration and deceleration. The values set with this instruction define the maximum acceleration and deceleration magnitudes that will be achieved.

When the V+ system is initialized, the profile, acceleration, and deceleration values are set to initial values, which can be defined by the SPEC utility program. As delivered by Adept, the selected profile is initially set to 1. The settings are not affected when program execution starts or stops, or when a ZERO command is processed.

Normally, the robot manufacturer sets the 100% acceleration and deceleration values to rates that can be achieved with typical payloads and robot link inertias. However, because the actual attainable accelerations vary greatly as a function of the end effector, payload, and the initial and final locations of a motion, accelerations greater than 100% may be permitted for your robot. The limits for the maximum values are defined by the robot manufacturer and vary from one type of robot to the next. If you specify a higher acceleration than is permitted, the limit established by the robot manufacturer is utilized.

You can use the functions ACCEL(3) and ACCEL(4) to determine the maximum allowable acceleration and deceleration settings.

For a given motion, the maximum attainable acceleration may actually be less than what you have requested. This occurs when a profile with a nonzero acceleration ramp time is used and there is insufficient time to ramp up to the maximum acceleration. That is, for a given jerk, a specific time must elapse before the acceleration can be changed from zero to the specified maximum value. If the maximum acceleration cannot be achieved, the trapezoidal profile will be reduced to a triangular shape. There are two circumstances when this occurs:

1. The motion is too short. In this case, the change in position is achieved before the maximum acceleration can be achieved.
2. The maximum motion speed is too low. In this case, the maximum speed is achieved before the maximum acceleration.

In both of these situations, raising the maximum acceleration and deceleration values does not affect the time for the motion.

Hint: If you increase the maximum acceleration and deceleration values but the motion time does not change, try the following: increase the program speed; switch to an acceleration profile that allows faster acceleration ramp times; or switch to acceleration profile 0, which specifies a square-wave acceleration profile.

> **NOTE:** This type of acceleration limiting cannot occur with acceleration profile 0 because a square-wave acceleration instantaneously changes acceleration values without ramping.

Examples

Set the default acceleration time to 50% of normal and the deceleration time to 30% of normal:

```
ACCEL 50, 30
```

Change the deceleration time to 60% of normal; leave acceleration alone:

```
ACCEL ,60
```

Reduce the acceleration and deceleration to one half of their current settings:

```
ACCEL ACCEL(1)/2, ACCEL(2)/2
```

Invoke defined profile #2 and set the acceleration magnitude to 80% of the defined rate:

```
ACCEL (2) 80
```

Related Keywords

ACCEL (real-valued function)

DURATION (program instruction)

PAYLOAD (program instruction)

SCALE.ACCEL (system switch)

SELECT (program instruction and real-valued function)

SPEED (monitor command and program instruction)

Syntax

ACCEL (*select*)

Function

Return the current setting for robot acceleration or deceleration setting or return the maximum allowable percentage limits set by the SPEC utility program (see the ***Instructions for Adept Utility Programs*** for details).

Usage Considerations

The ACCEL function returns information for the robot selected by the task executing the function.

If the V+ system is not configured to control a robot, use of the ACCEL function does not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

Parameter

select Real-valued expression, the result of which is rounded to an integer to select the value that is returned.

select	Value returned
0	Number of selected acceleration profile
1	Acceleration
2	Deceleration
3	Maximum allowable percentage acceleration
4	Maximum allowable percentage deceleration
5	Program speed below which acceleration and deceleration are scaled proportional to a program's speed setting when the SCALE.ACCEL system switch is enabled

Examples

```
ACCEL(1) ;Return the current acceleration setting.
ACCEL(2) ;Return the current deceleration setting.
```

Related Keywords

ACCEL (program instruction)
SCALE.ACCEL (system switch)
SELECT (program instruction and real-valued function)

Syntax

```
AIO.IN (channel, gain)
```

Function

Read a channel from one of the analog IO boards.

Usage Considerations

Analog IO must be installed in the system and correctly configured. See the controller user's guide for details on hardware configuration. See the *V+ Language User's Guide* for details on programming analog IO.

Parameters

channel Integer that specifies the analog IO board and channel to read. The valid ranges are 0, 1 to 32, and 1001 to 1256, inclusive (see Details). If an output channel number is specified, the last value written to that channel is returned. The special value 0 is used for testing.

gain Optional integer that specifies the gain for the channel. Must match the gain set by jumpers on the IO board (see the *Adept MV Controller User's Guide* for details). Acceptable values are 0 - 3. The default is 0.

Details

Each analog IO board contains 32 input (16 for differential input) and 4 output channels. A maximum of 8 boards can be installed. **Figure 2-2** shows the addressing for each board in the system.

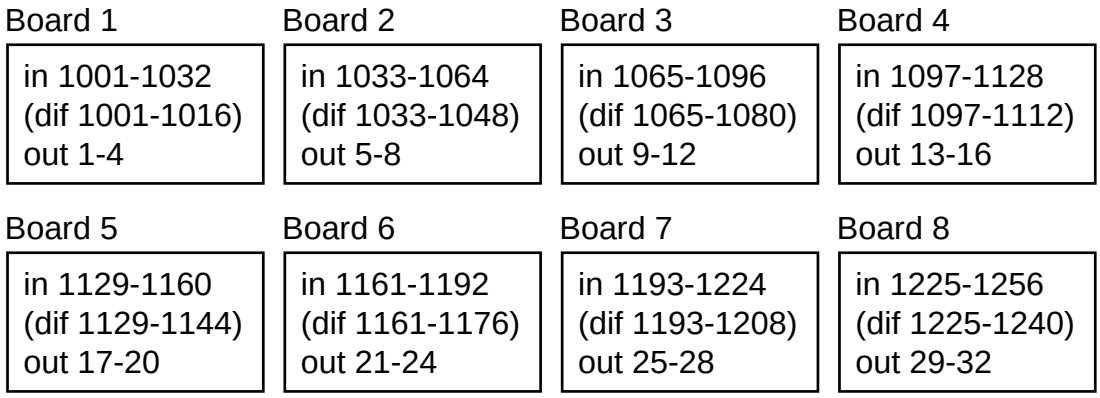


Figure 2-2. Analog Channel Addressing

The value returned by this function is in the range –1.0 to +1.0 (if the board is set for bipolar output), or 0 to 1.0 (if the board is set for unipolar output). Contact Adept applications for details on setting the configuration; refer to **"Application Questions" on page 38** for phone numbers.

Related Keywords

AIO.OUT (program instruction)

AIO.INS (real-valued function)

Syntax

```
AIO.INS (channel)
```

Function

Test whether an analog input or output channel is installed.

Parameter

channel Integer that specifies the analog IO board and channel to test. The valid ranges are 0, 1 to 32, and 1001 to 1256, inclusive.

Details

Each analog IO board contains 32 input (16 for differential input) and 4 output channels. A maximum of 8 boards can be installed. **Figure 2-2** shows the addressing for each board in the system.

TRUE is returned if the channel is installed or 0 is specified. FALSE is returned if the channel is not installed.

Related Keywords

AIO.IN (real-valued function)

AIO.OUT (program instruction)

Syntax

```
AIO.OUT channel = value
```

Function

Write to a channel on one of the analog IO boards.

Usage Considerations

Analog IO must be installed in the system and correctly configured. See the controller user's guide for details on hardware configuration. Contact Adept applications for details on programming analog IO; refer to "Application Questions" on page 38 for phone numbers.

Parameters

channel	Integer that specifies the analog IO board and channel to read. Acceptable values are 1 to 32 (see Figure 2-2 for details on channel addressing).
value	Value to output to the analog channel.

Details

Each analog IO board contains 32 input (16 for differential input) and 4 output channels. A maximum of 8 boards can be installed. Figure 2-2 shows the addressing for each board in the system.

The **value** parameter is assumed to be in the range –1.0 to +1.0 if the board is set for bipolar output, or 0 to 1.0 if the board is set for unipolar output (contact Adept applications for details). If value is outside this range, it will be clipped to be within the range. The actual voltage or current output by the analog I/O board is determined by jumper settings on the board (contact Adept applications for details).

Related Keywords

AIO.IN	(real-valued function)
AIO.INS	(real-valued function)

Syntax

ALIGN

Function

Align the robot tool Z axis with the nearest world axis.

Usage Considerations

The ALIGN instruction can be executed by any program task so long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the ALIGN instruction causes an error.

Details

Causes the tool to be rotated so that its Z axis is aligned parallel to the nearest axis of the World coordinate system. This instruction is primarily useful for lining up the tool before a series of locations are taught. This is most easily done by using the monitor DO command.

Related Keyword

SELECT (program instruction and real-valued function)

Syntax

```
ALTER (control) Dx, Dy, Dz, Rx, Ry, Rz
```

Function

Specify the magnitude of the real-time path modification that is to be applied to the robot path during the next trajectory computation.

Usage Considerations

This option is available only if your Adept system is equipped with the V+ Extensions option.

This instruction can be executed by the task that is controlling a robot in alter mode (see the **ALTON** instruction), or by any other task that has SELECTed the robot.

This instruction is ignored if the selected robot is not in alter mode.

When alter mode is enabled, this instruction should be executed once during each trajectory cycle (every 16 milliseconds). If this instruction is executed more often, only the last set of values defined during each cycle will be used.

Parameters

`control`	Optional real value, variable, or expression specifying a series of control bits. Currently, this parameter is not utilized, and it can be omitted.

`Dx`
`Dy`
`Dz`
`Rx`
`Ry`
`Rz`
	Optional real values, variables, or expressions that define the translations along, and the rotations about, the X, Y, and Z axes.
	
	In cumulative mode, omitted coordinates are interpreted as zero. In noncumulative mode, omitted coordinates default to the values specified in the previous ALTER instruction.
	
	Distances are interpreted as millimeters; angles are interpreted as degrees.

Details

After alter mode has been enabled (see the **ALTON** instruction), this instruction is executed once each trajectory-generation cycle to specify the amount by which the path is to be modified. The coordinates defined by this instruction are interpreted according to the modes specified by the ALTON instruction that initiated alter mode.

Example

The following pair of instructions could be embedded in a program loop that uses sensor data to control the motion of the robot.

```
; Modify the current robot path by (0.1*sx) millimeters
; along the X axis and (0.2*sz) millimeters along the Z axis,
; where "sx" and "sz" are variables that contain current
; sensor input data.

ALTER () 0.1*sx,,0.2*sz

; Wait for the next system cycle to give the trajectory
; generator a chance to execute.

WAIT
```

Related Keywords

ALTOFF (program instruction)

ALTON (program instruction)

STATE (real-valued function)

Syntax

ALTOFF

Function

Terminate real-time path-modification mode (alter mode).

Usage Considerations

This option is available only if your Adept system is equipped with the V+ Extensions option.

A robot must be attached by the program task prior to executing this instruction.

Turning off alter mode causes a BREAK in continuous-path motion.

Details

This instruction suspends program execution until any previous robot motion has been completed (similarly to the BREAK instruction), and then terminates real-time path-modification mode. After alter mode terminates, the robot is left at a final location that reflects both the destination of the last robot motion and the total ALTER correction that has been applied.

Related Keywords

ALTER (program instruction)

ALTON (program instruction)

STATE (real-valued function)

Syntax

```
ALTON (lun) mode
```

Function

Enable real-time path-modification mode (alter mode), and specify the way in which ALTER coordinate information will be interpreted.

Usage Considerations

This option is available only if your Adept system is equipped with the V+ Extensions option.

A robot must be attached by the program task prior to executing this instruction.

Alter mode cannot be active at the time this instruction is executed.

Any motions that are performed while alter mode is enabled must be of the straight-line motion type and cannot be specified relative to a conveyor belt.

Parameters

`lun` Optional real value, variable, or expression that specifies a communications channel associated with the ALTER operation. Currently, this parameter is not utilized, and it can be omitted (the parentheses are required, however).

`mode` Optional real value, variable, or expression that defines how path-modification data specified by subsequent ALTER instructions are to be interpreted. The mode value is interpreted as a sequence of bit flags, as detailed below. All the bits are assumed to be clear if the `mode` parameter is omitted.

Bit 1 (LSB) Accumulate corrections (mask value = 1)

If this bit is set, coordinate values specified by subsequent ALTER instructions are interpreted as incremental and are accumulated. If this bit is clear, each set of coordinate values is interpreted as the total (noncumulative) correction to be applied.

Bit 2 World coordinates (mask value = 2)

If this bit is set, coordinate values specified by subsequent ALTER instructions are interpreted to be in the World coordinate system. If this bit is cleared, coordinates are interpreted to be in the Tool coordinate system.

Details

This instruction initiates the real-time path-modification (alter) facility. After this instruction is executed, during all subsequent robot motions the coordinate values specified by ALTER instructions will automatically be superimposed every 16 milliseconds on the nominal path computed by the V+ trajectory generator. The corrections can be applied in all six degrees of freedom, and they can be specified as cumulative or noncumulative values in World or Tool coordinates.

Once alter mode is initiated, the robot location is corrected during all subsequent motions, and between motions if breaks occur between continuous-path segments. Alter mode is terminated by any of the following:

- Executing an ALTOFF instruction
- DETACHing the robot
- Prematurely terminating a robot motion
- Stopping program execution

Example

Initiate alter mode and interpret subsequent ALTER instructions as incremental (bit #1 is clear), World-coordinate (bit #2 is set) corrections to the nominal path of the robot:

```
ALTON () 2
```

Related Keywords

ALTER (program instruction)

ALTOFF (program instruction)

STATE (real-valued function)

Syntax

```
... ALWAYS
```

Function

Used with certain program instructions to specify a long-term effect.

Details

ALWAYS can be specified with any of the instructions listed below as related keywords. When ALWAYS is specified, the effect of the instruction continues until explicitly disabled. Otherwise, the effect of the instruction applies only to the next robot motion.

Examples

Permanently set the robot motion speed:

```
SPEED 50 ALWAYS
```

Permanently set loose-tolerance servo mode:

```
COARSE ALWAYS
```

Related Keywords

COARSE (program instruction)

DURATION (program instruction)

FINE (program instruction)

MULTIPLE (program instruction)

NONULL (program instruction)

NOOVERLAP (program instruction)

NULL (program instruction)

OVERLAP (program instruction)

SINGLE (program instruction)

SPEED (program instruction)

Syntax

```
... value AND value ...
```

Function

Perform the *logical* AND operation on two values.

Details

The AND operator operates on two values, resulting in their logical AND combination. For example, during the AND operation

```
c = a AND b
```

the following four situations can occur:

a	b		c
FALSE	FALSE	➡	FALSE
FALSE	TRUE	➡	FALSE
TRUE	FALSE	➡	FALSE
TRUE	TRUE	➡	TRUE

The result is TRUE only if *both* of the two operand values are logically TRUE.

Example

```
IF ready AND (count == 1) THEN

;The instructions following the IF will be executed if
;both "ready" is TRUE (nonzero) and "count" equals 1.
```

Related Keywords

BAND (operator)

OR (operator)

XOR (operator)

Syntax

ANY

Function

Signal the beginning of an alternative group of instructions for the CASE structure.

Usage Considerations

The ANY instruction must be within a CASE structure.

Details

See the description of the **CASE** structure.

Related Keywords

CASE (program instruction)

VALUE (program instruction)

Syntax

```
APPRO location, distance

APPROS location, distance
```

Function

Start a robot motion toward a location defined relative to specified location.

Usage Considerations

APPRO causes a joint-interpolated motion.

APPROS causes a straight-line motion, during which no changes in configuration are permitted.

The APPRO and APPROS instructions can be executed by any program task so long as the task has attached a robot. The instructions apply to the robot selected by the task.

If the V+ system is not configured to control a robot, executing these instructions will cause an error.

Parameters

`location` Transformation value that defines the basis for the final location.

`distance` Real-valued expression that specifies the distance along the robot tool Z axis between the specified location and the actual desired destination.

A positive distance sets the tool back (negative tool-Z) from the specified location; a negative distance offsets the tool forward (positive tool-Z).

Details

These instructions initiate a robot motion to the orientation described by the given location value. The position of the destination location is offset from the given location by the distance given, measured along the tool Z axis.

Examples

```
APPRO place,offset
```

Moves the tool, by joint-interpolated motion, to a location offset millimeters from that defined by the transformation place. The offset is along the resultant Z axis of the tool.

```
APPROS place,-50
```

Moves the tool along a straight line to a location 50 millimeters from that defined by the transformation place, with the offset along the resultant Z axis of the tool to a location **beyond** the location place.

Related Keywords

DEPART and **DEPARTS** (program instructions)

MOVEF and **MOVESF** (program instructions)

SELECT (program instruction and real-valued function)

Syntax

```
ASC (string, index)
```

Function

Return an ASCII character value from within a string.

Parameters

`string` String expression from which the character is to be picked. If the string is empty, the function returns the value –1.

`index` Optional real-valued expression defining the character position of interest. The first character of the string is selected if the index is omitted or has a value of 0 or 1.

If the value of the index is negative, or greater than the length of the string, the function returns the value –1.

Details

The ASCII value of the selected character is returned as a real value.

Examples

```
ASC("sample", 2)
;Returns the ASCII value of the letter "a".

ASC($name)
;Returns the ASCII value of the first character of the
;string contained in the variable $name.

ASC($system, i)
;Uses the value of the real variable "i" as an index to
;the character of interest in the string contained in the
;variable "$system".
```

Related Keywords

$CHR (string function)

VAL (real-valued function)

Syntax

ATAN2 (value_1, value_2)

Function

Return the size of the angle (in degrees) that has its trigonometric tangent equal to value_1/value_2.

Usage Considerations

The returned value is zero if both parameter values are zero.

Parameters

value_1 Real-valued expression.

value_2 Real-valued expression.

Examples

```
ATAN2(0.123,0.251)      ;Returns  26.1067
ATAN2(–5.462,47.2)      ;Returns  -6.600928
ATAN2(1.3125E+2,–1.3)   ;Returns  -90.56748
slope = ATAN2(rise, run)
```

NOTE: TYPE, PROMPT, and similar instructions output the results of the above examples as single-precision values. However, they are actually stored and manipulated as double-precision values. The LISTR monitor command will display real values to full precision.

Syntax

```
ATTACH (lun, mode) $device
```

Function

Make a device available for use by the application program.

Usage Considerations

The robot is automatically attached when the EXECUTE monitor command or program instruction is processed (with the DRY.RUN system switch disabled). All the other logical units are automatically detached when program execution begins.

The logical unit for the manual control pendant can be attached even when the pendant is not connected to the system. The function PENDANT(-4) can be used (before or after attaching) to determine if the pendant is connected.

If the system terminal or the manual control pendant was attached when a program stopped executing, it is automatically reattached if execution of the program is resumed with the PROCEED, RETRY, SSTEP, or XSTEP commands.

Use of this instruction to attach to NFS or TCP devices requires the AdeptNet option and the appropriate license(s).

Parameters

`lun` The logical unit number to associate with the attached device. The interpretation of this parameter depends on the value of the `mode` parameter, as follows:

If bit 3 of the `mode` parameter is 0, this parameter is optional (defaulting to 0, to attach the robot); and it can be a real value, variable, or expression (interpreted as an integer) in the range 0 to 24 that *specifies* the logical unit to be attached. See the Details section for the default association of logical units with devices. If the logical unit specified is not 0, you can use the `$device` parameter to override the default device for the logical unit.

If bit 3 of the `mode` parameter is 1, this parameter is required and must be a real variable. In this case, the V+ system automatically assigns a logical unit to the device specified by the `$device` parameter and to this parameter. The parameter is set to –1 if all the logical units are in use. V+ assigns a value to the `lun` parameter even if the ATTACH request fails.

`mode` Optional real value, variable, or expression (interpreted as a bit field) that defines how the ATTACH request is to be processed. The value specified is interpreted as a sequence of bit flags as detailed below. All the bits are assumed to be clear if no value is specified.)

Bit 1 (LSB) Queue (0) versus Fail (1) (mask value = 1)

This bit controls how the device driver responds to the attach request from the control program task. (The device driver is an internal system task that is separate from the control program task.) For most applications, this bit should be set.

If this bit is clear, and the device is already attached by another control program task, the driver queues this attach request and signals the control program that the attach is not complete. The attachment will complete when the device becomes available.

If this bit is set, and the device is already attached by another control program task, the device driver immediately signals that the attach request has failed.

The function IOSTAT(lun) can be used to determine the success or failure of the attachment. A positive value from IOSTAT indicates successful completion; zero indicates the attachment has not completed; a negative value indicates completion with an error.

Bit 2 Wait (0) versus No-wait (1) (mask value = 2)

This bit controls whether or not the control program task waits for a response from the device driver. For most applications, this bit should not be set.

If this bit is clear, program execution waits for the device driver to signal the result of the attach request.

If this bit is set, program execution does not wait for the result of the attach request. The program must then use the function IOSTAT(lun) to determine if the attachment has succeeded (see earlier text). If the program attempts to READ from or WRITE to the logical unit while the attachment is pending, program execution then waits for the attachment to complete.

Bit 3 Specify LUN (0) vs. Have LUN assigned (1) (mask value = 4)

This bit determines how the `lun` parameter is processed.

If this bit is clear, the device corresponding to the value of `lun` is attached. That is, the value of the `lun` parameter specifies the device that is to be attached (according to the table in the Details section) except when a different device is specified with the `$device` parameter.

If this bit is set, the device to be attached is specified by the `$device` parameter (which should not be omitted). In this case, a logical unit is automatically selected, and the value of the `lun` parameter is *set* by the ATTACH instruction. V+ assigns a value to `lun` even if the ATTACH request fails. (This mode cannot be used to attach the robot or manual control pendant.)

`$device` Optional string constant, variable, or expression that identifies the device to be attached. If bit 3 of the `mode` parameter is 0, this parameter is used to override the default device associated with the value of the `lun` parameter (except that logical unit 0 is always the robot).

The acceptable device names are shown in **Table 2-1 on page 72**.

Table 2-1. Acceptable Device Names to be Attached

Device	Meaning
DDCMP:n	Global serial line (see SERIAL:n below) configured for DDCMP support[a]
DISK	Physical disk
GRAPHICS	Graphics window
KERMIT	Serial line that is configured for Kermit support[b]
LOCAL.SERIAL:n	Local serial line (n = 1 or 2 for Adept CPUs)[c]
MONITOR	The current monitor window or operator's terminal
NFS	NFS protocol device driver
SERIAL:n	Global serial line (n = 1, 2, 3, or 4 for Adept SIO board—manual control pendant uses global serial line 4)[c]
SYSTEM	Disk device, drive, and subdirectory path currently set with the DEFAULT command
TCP	TCP protocol device driver

[a] The global serial line used for DDCMP must have been configured for DDCMP protocol using CONFIG_C

[b] The global or local serial line used for KERMIT must have been configured for KERMIT protocol using CONFIG_C

[c] The initial settings for these serial lines are set with CONFIG_C. They may be changed with the FSET program instruction or monitor command.

Details

The robot must remain attached by a robot control program for the program to command motion of the robot. When the robot is detached (see the DETACH instruction), however, the operator can use the manual control pendant to move the robot under directions from the application program. This is useful, for example, for application setup sequences. (The belt and vision calibration programs provided by Adept use this technique.)

Program task 0 automatically attaches robot #1 when that task begins execution. A robot control program executed by any of the other program tasks must explicitly attach the robot.

Any task can attach to any robot, provided that the robot is not already attached to a different task. The robot that is attached by an ATTACH instruction is the one that was last specified by a SELECT instruction executed by the current task (see the SELECT instruction). If no SELECT instruction has been executed, then robot #1 is attached. The SELECT instruction can be used to select a different robot only if no robot is currently attached to the task.

To successfully attach the robot, the system must be in COMP mode. Otherwise (for `mode` bit 1 = 0), program execution is suspended (without notice) until the system is placed in COMP mode. This situation can be avoided in two ways: (1) use the STATE function to determine if the system is in COMP mode before executing an ATTACH instruction, (2) set bit 1 in the `mode` value, and use the IOSTAT function to determine the success of the ATTACH instruction.

The manual control pendant (logical unit 1) must be attached before (1) text can be sent to the display, (2) a KEYMODE instruction can be processed, or (3) keys can be read in keyboard mode or special mode (see KEYMODE). When the manual control pendant is attached, the USER light on the pendant is turned on. The light will blink if the pendant is not in background mode (for example, it is in DISP mode) while an input or output operation is pending.

> **NOTE:** The logical unit for the manual control pendant can be attached even when the pendant is not connected to the system. The function PENDANT(–4) can be used (before or after attaching) to determine if the pendant is connected.

When the system terminal (logical unit 4) is attached, all keyboard input will be buffered for input requests by the program.

> **NOTE:** When the system terminal is attached, the user will not be able to type ABORT to terminate program execution. The program will have to provide a means for fielding a termination request, or the user will have to use the manual control pendant or emergency stop switch to stop program execution.

When a DISK is attached, a program can write and read data to and from the local disk drives, and to and from remote systems via the Kermit protocol. One of the FOPEN_ instructions must be used to specify which disk to use, and which file on the disk. WRITE and READ instructions can then be used to transfer information to and from the disk. Also, FCMND instructions can be used to send commands to the disk.

When a LOCAL.SERIAL:n or SERIAL:n serial communication line is attached, it can be used to send and receive information to and from another system. As with the disk, WRITE and READ instructions are used for the information transfer. When a serial communication line is attached, its configuration is set to that on the boot disk from which the V+ system was loaded. That is, ATTACH cancels the effect of any FSET that had been used previously to modify the characteristics of the serial line.

When a GRAPHICS logical unit is attached, a program can access graphics windows on the system monitor (A-series and AdeptWindowsPC systems only). After a window logical unit is attached, the FOPEN instruction must be used to specify which window will be accessed with the logical unit. The FSET instruction can then be used to modify attributes of the window, the graphics instructions can be used to output to the window, and the GETEVENT instruction can be used to receive input from the window.

When `mode` bit 3 = 0 and the `$device` parameter is omitted, the logical unit number implicitly specifies the corresponding default device from **Table 2-2**.

Table 2-2. Default Device Numbers Supplied by the LUN

Number	Device
0	Robot (default when `lun` is omitted)
1	Manual control pendant
2	System terminal
3	System terminal
4	System terminal
5	Disk
6	Disk
7	Disk
8	Disk
9	No default device
10	Serial communication line (Global #1 on SIO board)

Table 2-2. Default Device Numbers Supplied by the LUN (Continued)

Number	Device
11	Serial communication line (Global #2 on SIO board)
12	Serial communication line (Global #3 on SIO board)
13	Serial communication line (Global #4 on SIO board)
14	Serial communication line (Local #1 on system processor)
15	Serial communication line (Local #2 on system processor)
16	No default device
17	Disk
18	Disk
19	Disk
20	Graphics window
21	Graphics window
22	Graphics window
23	Graphics window
24	Graphics window
25 - 31	No default devices

Logical units 5 to 8 and 17 to 19 can also be used for disk-like devices such as KERMIT or remote disks on a network.

Examples

Take over control of the robot:

```
ATTACH
```

Connect to global serial line 1; wait for it to become available if another task has it attached; return the assigned logical unit number in `lun` :

```
ATTACH (lun, 4) "serial:1"
```

The next instruction is very similar to the previous one, but this one requires use of the IOSTAT function to determine if another task has the serial line attached:

```
ATTACH (lun, 5) "serial:1"
```

Attach to the TCP device driver with automatic allocation of a logical unit number:

```
ATTACH (lun, 4) "TCP"
```

Attach to the NFS device driver with automatic allocation of a logical unit number:

```
ATTACH (lun, 4) "NFS"
```

Related Keywords

DETACH (program instruction)

FSET (program instruction)

IOSTAT (real-valued function)

SELECT (program instruction and real-valued function)

Syntax

```
AUTO type variable, ..., variable
```

Function

Declare temporary variables that are automatically created on the program stack when the program is entered.

Usage Considerations

AUTOmatic variables have an undetermined value when a program is first entered (but they are *not* necessarily undefined), and they have no value after the program exits.

AUTO statements must appear before any executable instruction in the program—only the .PROGRAM statement, comments, blank lines, GLOBAL and LOCAL statements, and other AUTO statements may precede this instruction.

If a variable is listed in an AUTO statement, any global variables with the same name cannot be accessed directly by the program.

The values of AUTOmatic variables are not saved by the STORE or restored by the LOAD monitor commands.

Parameters

`type` — Optional keyword REAL, DOUBLE, or LOC, indicating that all the variables in this statement are to be single precision, double precision, or location variables. (A location can be a transformation, precision point, or belt variable.)

If this keyword is omitted, the type of each variable is determined by its use within the program. An error is generated if the type cannot be determined from usage.

`variable` — Name of a variable of any data type available with V+ (belt, precision point, real-value, string, and transformation). Each variable can be a simple variable or an array. If the `type` parameter is specified (see below), all the variables must match that type. Array variables must have their indexes specified explicitly, indicating the highest valid index for the array.

Details

This instruction is used to declare variables to be defined only within the current program. That is, an AUTOmatic variable can be referenced only by the specific calling instance of a program. Also, the names of AUTOmatic variables can be selected without regard for the names of variables defined in any other programs (except for global variables; see following text).

AUTOmatic variables are allocated each time the program is called, and their values are not preserved between successive subroutine calls. These values can be displayed via monitor commands only when the program task is inactive but is on an execution stack. When a program is first entered, automatic variables have arbitrary, undetermined values (and they are *not* necessarily undefined). AUTOmatic variables are lost when the program exits.

Unlike a LOCAL variable, a separate copy of an AUTOmatic variable is created *each time* a program is called, even if it is called simultaneously by several different program tasks, or called recursively by a single task. If a program that uses LOCAL or global variables is called by several different program tasks, or recursively by a single task, the values of those variables can be modified by the different program instances and can cause very strange program errors. Therefore, AUTOmatic variables should be used for all temporary local variables to minimize the chance of such errors.

Variables can be defined as GLOBAL, AUTOmatic, or LOCAL. An attempt to define AUTOmatic, GLOBAL, or LOCAL variables with the same name will result in the error message `*Attempt to redefine variable class*`.

Variables can be defined only once within the same context (AUTOmatic, LOCAL, or GLOBAL). Attempting to define a variable more than once (that is, with a different type) will yield the error message `*Attempt to redefine variable type*`.

AUTOmatic array variables must have the size of each dimension specified in the AUTO statement. Each index specified must represent the last element to be referenced in that dimension. The first element allocated always has index value zero. For example, the statement

```
AUTO LOC points[3,5]
```

allocates a transformation array with 24 elements. The left-hand index ranges from 0 to 3, and the right-hand index ranges from 0 to 5.

The storage space for AUTOmatic variables is allocated on the program execution stack. If the stack is too small for the number of AUTOmatic variables declared, the task execution will stop with the error message

```
*Too many subroutine calls*
```

If this happens, use the STACK monitor command to increase the stack size and then issue the RETRY monitor command to continue program execution.

AUTOmatic variables cannot be deleted with the DELETE_ commands.

AUTOmatic variables can be referenced with monitor commands such as BPT, DELETE_, DO, HERE, LIST_, POINT, TEACH, TOOL, and WATCH by using the optional context specifier @. The general syntax is:

```
command @task:program command_arguments
```

Examples

Declare the variables **loc.a**, **$ans**, and **i** to be AUTOmatic in the current program (the variable types for **loc.a** and **i** must be clear from their use in the program):

```
AUTO loc.a, $ans, i
```

Declare the variables **i**, **j**, and **tmp[ ]** to be AUTOmatic, real variables in the current program (array elements **tmp[0]** through **tmp[10]** are defined):

```
AUTO REAL i, j, tmp[10]
```

Declare the variable **loc** to be an AUTOmatic variable in the current program. The variable type of **loc** must be determined by its use in the program. Note that since LOC appears by itself, it is *not* interpreted as the type-specifying keyword.)

```
AUTO loc
```

Related Keywords

GLOBAL	(program instruction)
LOCAL	(program instruction)
STACK	(monitor command, see the ***V+ Operating System Reference Guide)***

Syntax

```
AUTO.POWER.OFF
```

Function

Control whether or not V+ disables high power when certain motion errors occur.

Usage Considerations

This switch has effect during automatic mode but not during manual mode. It is especially useful in reducing operator intervention during common nulling-timeout and envelope errors.

Details

Because the HIGH POWER ON/OFF button cannot be used by itself to enable high power as in earlier versions of V+, Adept has sought to reduce the number of instances that high power is disabled during normal program execution. Making this improvement allows programs to continue to recover automatically from errors without manual intervention, that is, without requiring the operator to press the HIGH POWER ON/OFF button. This system switch cancels the effect of this change. By default this switch is disabled. Enabling it restores functionality as it was in V+ version 12.1 and earlier.

Adept reviewed all automatic-mode errors that disabled high power in V+ version 12.1 and determined which could be changed simply to decelerate the robot and generate an error without compromising the safe operation of the system. Examples of particular importance are errors such as nulling-timeout and envelope errors that often occur during the normal operation of the system. In some cases, Adept has modified internal software to ensure the continued safe operation of your system.

The setting of this switch has no effect during manual mode.

If this switch is set to nonzero, V+ does not report any servo 3 errors. That is, all the servo errors disable high power.

Example

The following program segment instructs V+ to disable high power when any motion error occurs:

```
ENABLE AUTO.POWER.OFF   ;Disable high power when any
                        ;motion error occurs
```

Related Keywords

DISABLE (monitor command and program instruction)

ENABLE (monitor command and program instruction)

SWITCH (monitor command, program instruction, and real-valued function)

Syntax

```
... value BAND value ...
```

Function

Perform the **binary** AND operation on two values.

The BAND operation is meaningful only when performed on integer values.

Details

The BAND operator can be used to perform a binary AND operation on two values on a bit-by-bit basis, resulting in a real value.

Specifically, the BAND operation consists of the following steps:

1. Convert the operands to sign-extended 24-bit integers, truncating any fractional part.
2. Perform a binary AND operation (see below).
3. Convert the result back to a real value.

During the binary AND operation,

```
c = a BAND b
```

the bits in the resultant C are determined by comparing the corresponding bits in the operands A and B as indicated below.

Table 2-3. Converting A and B Operands into Real Values (BAND Operator)

For each bit in: a	b		c
0	0	➡	0
0	1	➡	0
1	0	➡	0
1	1	➡	1

That is, a bit in the result will be 1 if the corresponding bit in both of the operands is 1.

Refer to the *V+ Language User's Guide* for the order in which operators are evaluated within expressions.

Examples

Consider the following (binary values are used to make the operation more evident):

```
^B101000   BAND  ^B100001  yields  ^B100000  (32)
```

Note that a very different result is obtained with the logical AND operation:

```
^B101000   AND  ^B100001  yields  -1  (TRUE)
```

In this case, ^B101000 and ^B100001 are each interpreted as logically TRUE since they are nonzero.

Related Keywords

AND (operator)

BOR (operator)

BXOR (operator)

Syntax

```
BASE X_shift, Y_shift, Z_shift, Z_rotation
```

Function

Translate and rotate the World reference frame relative to the robot.

Usage Considerations

The BASE program instruction causes a BREAK in continuous-path motion.

The BASE monitor command applies to the robot selected by the V+ monitor (with the SELECT command). The command can be used while programs are executing. However, an error will result if the robot is attached by any executing program.

The BASE instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, use of the BASE command or instruction will cause an error.

Parameters

`X_shift`	Optional real-valued expression describing the X component (in the normal World coordinate system) of the origin point for the new coordinate system. (Zero is assumed if no value is provided.)
`Y_shift`	Similar to X_shift, but for the Y direction.
`Z_shift`	Similar to X_shift, but for the Z direction.
`Z_rotation`	Similar to X_shift, but for a rotation about the Z axis.

Details

When the V+ system is initialized, the origin of the reference frame of the robot is assumed to be fixed in space such that the X-Y plane is at the robot mounting surface, the X axis is in the direction defined by joint 1 equal to zero, and the Z axis coincides with the joint-1 axis.

The BASE instruction offsets and rotates the reference frame as specified above. This is useful if the robot is moved after locations have been defined for an application.

If, after robot locations have been defined by transformations relative to the robot reference frame, the robot is moved relative to those locations—to a point translated by <dX>,<dY>,<dZ> and rotated by <Z rotation> degrees about the Z axis—a BASE command or instruction can be used to compensate so that motions to the previously defined locations will still be as desired.

Another convenient use for the BASE command or instruction is to realign the X and Y coordinate axes so that SHIFT functions cause displacements in desired, nonstandard directions.

> **NOTE:** The BASE instruction has no effect on locations defined as precision points. The arguments for the BASE instruction describe the displacement of the *robot* relative to its normal location.
>
> The BASE function can be used with the LISTL command to display the current BASE setting.

Examples

```
BASE xbase,, -50.5, 30
```

Redefines the World reference frame because the robot has been shifted xbase millimeters in the positive X direction and 50.5 millimeters in the negative Z direction, and has been rotated 30 degrees about the Z axis.

```
BASE 100,, -50
```

Redefines the World reference frame to effectively shift all locations 100 millimeters in the negative X direction and 50 millimeters in the positive Z direction from their nominal location. Note that the arguments for this instruction describe movement of the robot reference frame relative to the robot, and thus have an opposite effect on locations relative to the robot.

Related Keywords

BASE	(transformation function)
SELECT	(monitor command, program instruction, and real-valued function)

Syntax

BASE

Function

Return the transformation value that represents the translation and rotation set by the last BASE command or instruction.

Usage Considerations

The BASE function returns information for the robot selected by the task executing the function.

The command LISTL BASE can be used to display the current base setting.

If the V+ system is not configured to control a robot, use of the BASE function will not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

Related Keywords

BASE (monitor command and program instruction)

SELECT (program instruction and real-valued function)

Syntax

BCD (value)

Function

Convert a real value to Binary Coded Decimal (BCD) format.

Usage Considerations

The BCD function is most useful when used in conjunction with the BITS command, instruction, and function (see below).

Parameter

value Real-valued expression defining the value to be converted.

Details

The BCD function converts an integer value in the range 0 to 9999 into its BCD representation. This can be used to set a BCD value on a set of external output signals.

Example

If you want to use digital signals 4 to 8 to output a BCD digit: The instruction

```
BITS 4,4 = BCD(digit)
```

would convert the value of the real variable digit to BCD and impress it on external output signals 4-8.

Related Keyword

DCB (real-valued function)

Syntax

BELOW

Function

Request a change in the robot configuration during the next motion so that the elbow is below the line from the shoulder to the wrist.

Usage Considerations

Configuration changes cannot be made during straight-line motions.

If the selected robot does not support a below configuration, this instruction is ignored by the robot. (SCARA robots, for example, cannot be in an ABOVE/BELOW configuration.)

The BELOW instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the BELOW instruction will cause an error.

See **Figure 2-1 on page 46**.

Related Keywords

ABOVE	(program instruction)
CONFIG	(real-valued function)
SELECT	(program instruction and real-valued function)

Syntax

```
... BELT
```

Function

Control the function of the conveyor tracking features of the V+ system.

Usage Considerations

This option is available only if your Adept system is equipped with the V+ Extensions option.

If the V+ system is not configured to control a robot, an attempt to enable the BELT system switch will cause an error. (The DEVICE real-valued function and the SETDEVICE program instruction must be used to access external encoders from a nonrobot system. See the ***V+ Language User's Guide*** for more information.)

Details

This switch must be enabled before any of the special conveyor tracking instructions can be executed. When BELT is disabled, the conveyor tracking software has a minimal impact on the overall performance of the system.

When the BELT switch is enabled, error checking is initiated for the encoders associated with any belt variables that are defined. The switch is *disabled* when the V+ system is initialized.

Related Keywords

BELT	(real-valued function)
BELT.MODE	(system parameter)
BSTATUS	(real-valued function)
DEFBELT	(program instruction)
DISABLE	(monitor command and program instruction)
ENABLE	(monitor command and program instruction)
SETBELT	(program instruction)
SWITCH	(monitor command, program instruction, and real-valued function)
WINDOW	(program instruction and real-valued function)

Syntax

BELT (%belt_var, mode**)**

Function

Return information about a conveyor belt being tracked with the conveyor tracking feature.

Usage Considerations

This option is available only if your Adept system is equipped with the V+ Extensions option.

The BELT system switch must be enabled before this function can be used.

Parameters

%belt_var The name of the belt variable used to reference the conveyor belt. As with all belt variables, the name must begin with a percent symbol (%).

mode Control value that determines the information that will be returned.

If the mode is omitted or its value is less than or equal to zero, the BELT function returns the encoder reading of the belt specified by the belt variable.

If the value of the expression is greater than zero, the encoder velocity is returned in units of encoder counts per V+ cycle (16 ms).

Examples

Set the point of interest on the referenced conveyor to be that corresponding to the current reading of the belt encoder:

```
SETBELT %main.belt = BELT(%main.belt)
```

Save the current speed of the belt associated with the belt variable %b:

```
belt.speed = BELT(%b, 1)
```

Related Keywords

BELT (system switch)

SETBELT (program instruction)

Syntax

```
... BELT.MODE
```

Function

Set characteristics of the conveyor tracking feature of the V+ system.

Usage Considerations

This option is available only if your Adept system is equipped with the V+ Extensions option.

The current value of the BELT.MODE parameter can be determined with the PARAMETER monitor command or real-valued function.

The value of the BELT.MODE parameter can be modified only with the PARAMETER monitor command or program instruction.

Details

This parameter is interpreted as a bit-flag word. The initial setting of this parameter is 0. That is, all the bits are zero. Bits can be set by assigning the value resulting from adding together the desired bit mask values (see the example below).

The bit flags have the following interpretations:

Bit 1 (LSB) Upstream/downstream definition (mask value = 1)

When this bit is set to one, the instantaneous direction of travel of the belt is used to define upstream and downstream for the window testing routines (both in the internal motion planner and the WINDOW real-valued function).

When this bit is set to zero, going from upstream to downstream always corresponds to traveling in the direction of the positive X axis of the nominal transformation.

Bit 2 Stopped-belt processing (mask value = 2)

When this bit is set to one, a program error will be generated during motion planning if the destination is outside of the belt window and the belt is stopped.

When this bit is set to zero, if the belt is stopped during motion planning, the direction of the positive X axis of the nominal

transformation is used to define the downstream direction. The normal window-error criteria are then applied (see below).

Bit 3 — Window error definition (mask value = 4)

When this bit is set to one, destination locations that are downstream or upstream of the belt window cause motion instructions to fail during planning.

When this bit is set to zero, upstream window violations cause planning to wait until the location comes into the window. Destination locations that are downstream of the belt window cause window errors.

Bit 4 — Effect of window errors (mask value = 8)

When this bit is set to one, motion instructions that fail during planning due to a window error are ignored (skipped) and program execution continues as usual. When this option is selected, each belt-relative motion instruction should be followed by an explicit test for planning errors using the BSTATUS function.

When this bit is zero, window errors during motion planning generate a program step execution error, which either halts program execution or triggers the REACTE routine.

Regardless of the setting of this bit, window errors that occur while the robot is actually tracking the belt cause the program specified in the latest WINDOW instruction to be executed. If no such program has been specified, program execution is halted.

Example

Set the parameter to have bits 1 and 3 set to one (mask values 1 + 4):

```
PARAMETER BELT.MODE = 5
```

Related Keywords

BELT (system switch and real-valued function)

BSTATUS (real-valued function)

PARAMETER (monitor command, program instruction, and real-valued function)

WINDOW (program instruction and real-valued function)

Syntax

```
BITS first_sig, num_sigs = value
```

Function

Set or clear a group of digital signals based on a value.

Usage Considerations

Both external digital output signals and internal software signals can be referenced. Input signals must not be referenced. (Input signals are displayed by the monitor command IO 1.)

No more than eight signals can be set at one time.

Any group of up to eight signals can be set, providing that all the signals in the group are configured for use by the system.

Note that software signal 2032 (brake solenoid) is a read-only signal. Attempting to set this signal will result in an `*Illegal digital signal*` error message.

Parameters

`first_sig`	Real-valued expression defining the lowest-numbered signal to be affected.
`num_sigs`	Optional real-valued expression specifying the number of signals to be affected. A value of 1 is assumed if none is specified. The maximum valid value is 8.
`value`	Real-valued expression defining the value to be set on the specified signals. If the binary representation of the value has more bits than `num_sigs`, only the lowest `num_sigs` signals will be affected.

Details

Sets or clears one or more external output signals or internal software signals based on the value to the right of the equal sign. The effect of this instruction is to round **`value`** to an integer, and then set or clear a number of signals based on the individual bits of the binary representation of the integer.

Examples

Set external output signals 1-8 (8 bits) to the binary representation of the current monitor speed setting:

```
BITS 1,8 = SPEED(1)
```

If the monitor speed were currently set to 50% (0011 0010 binary), then signals 1-8 would be set as follows after this instruction:

signal 1 ➡ 0 (off) signal 5 ➡ 1(on)

signal 2 ➡ 1 (on) signal 6 ➡ 1 (on)

signal 3 ➡ 0 (off) signal 7 ➡ 0 (off)

signal 4 ➡ 0 (off) signal 8 ➡ 0 (off)

Set external output signals 5-9 (4 bits) to the binary representation of the BCD digit 7:

```
BITS 5,4 = BCD(7)
```

Set external output signals 1-8 (8 bits) to the binary representation of the constant 255, which is 11111111 (binary). Thus, signals 1-8 will all be turned on:

```
BITS 1,8 = 255
```

Related Keywords

BITS (real-valued function)

IO (monitor command, see the *V+Operating System Reference Guide*)

RESET (monitor command, see the *V+Operating System Reference Guide*)

SIG (real-valued function)

SIG.INS (real-valued function)

SIGNAL (monitor command and program instruction)

Syntax

BITS (first_sig, num_sigs**)**

Function

Read multiple digital signals and return the value corresponding to the binary bit pattern present on the signals.

Usage Considerations

External digital input or output signals, or internal software signals can be referenced.

A maximum of 8 signals can be read at one time.

Any group of up to eight signals can be read, providing that all the signals in the group are configured for use by the system.

Parameters

first_sig	Real-valued expression defining the lowest-numbered signal to be read.
num_sigs	Optional real-valued expression specifying the number of signals to be affected. A value of 1 is assumed if none is specified. The maximum valid value is 8.

Details

This function returns a value that corresponds to the binary bit pattern present on one to eight digital signals.

The binary representation of the value returned by the function has its least-significant bit determined by signal numbered **first_sig**, and its higher-order bits determined by the next num_sigs –1 signals.

Example

Assume that the following input signal states are present:

```
Signal:10081007100610051004100310021001
State: 1101011 0
```

The program step:

```
x = BITS(1003, 4)
```

will yield a value of 5 for x since the 4 signals starting at 1003 (that is, signals 1003 through 1006) can be interpreted as a binary representation of that value.

Related Keywords

BITS	(monitor command and program instruction)
IO	(monitor command, see the *V+Operating System Reference Guide*)
RESET	(monitor command, see the *V+Operating System Reference Guide*)
SIG	(real-valued function)
SIG.INS	(real-valued function)
SIGNAL	(monitor command and program instruction)

Syntax

BMASK (bit, bit, ..., bit**)**

Function

Create a bit mask by setting individual bits.

Parameter

bit	Integer value from 1 to 32 specifying a bit to turn on. The least-significant bit is number 1.

Details

This instruction creates a bit mask by turning on (bit = 1) the specified bits and leaving all other bits off (bit = 0).

Bit 32 is the sign bit and yields a negative number when set.

Examples

Create the bit mask ^B10001:

```
bm = BMASK(1, 5)
```

Attach to a disk LUN with mode bit 2 turned on:

```
mode = BMASK(2)
ATTACH (lun, mode) "DISK"
```

Syntax

```
... value BOR value ...
```

Function

Perform the binary OR operation on two values.

Usage Considerations

The BOR operation is meaningful only when performed on integer values.

Details

The BOR operator can be used to perform a binary OR operation on two values on a bit-by-bit basis, resulting in a real value. Note, however, that this operation is meaningful only when performed on integer values.

Specifically, the BOR operation consists of the following steps:

1. Convert the operands to sign-extended 24-bit integers, truncating any fractional part.
2. Perform a binary OR operation (see below).
3. Convert the result back to a real value.

During the binary OR operation,

```
c = a BOR b
```

the bits in the resultant C statement are determined by comparing the corresponding bits in the operands A and B as indicated in **Table 2-4**.

Table 2-4. Converting A and B Operands into Real Values (BOR Real-Valued Function)

For each bit in: a	b		c
0	0	➡	0
0	1	➡	1
1	0	➡	1
1	1	➡	1

That is, a bit in the result will be 1 if the corresponding bit in either of the operands is 1.

Refer to the *V+ Language User's Guide* for the order in which operators are evaluated within expressions.

Examples

Consider the following (binary values are shown only to make the operation more evident):

^B101000 BOR ^B100001 yields ^B101001 (41.0)

Note that a very different result is obtained with the logical OR operation:

^B101000 OR ^B100001 yields –1 (TRUE)

In this case, ^B101000 and ^B100001 are each interpreted as logically TRUE since they are nonzero.

Related Keywords

BAND (operator)

BXOR (operator)

OR (operator)

Syntax

BRAKE

Function

Abort the current robot motion.

Usage Considerations

The BRAKE instruction can be executed by any program task, including a task that is not actively controlling the robot.

This instruction does *not* cause a BREAK to occur (see Details below).

If more than one robot is connected to the controller, this instruction applies to the robot currently selected (see the **SELECT** instruction).

If the V+ system is not configured to control a robot, the BRAKE instruction will not generate an error due to the absence of a robot.

Details

BRAKE causes the current robot motion to be aborted immediately. In response to this instruction, the robot will decelerate to a stop and then (without waiting for position errors to null) begin the next motion.

Note, however, that program execution is **not** suspended until the robot motion stops.

Example

The following program segment initiates a robot motion and simultaneously tests for a condition to be met. If the condition is met, the motion is stopped with a BRAKE instruction. Otherwise, the motion is completed normally.

```
MOVES step[1]          ;Initiate motion to next location
DO                     ;Loop continuously...
   IF SIG(1023) THEN   ;If input signal 1023 becomes set,
     BRAKE             ;stop the motion immediately
     EXIT              ;and continue elsewhere
   END
UNTIL STATE(2) == 2    ;...until location reached
MOVES step[2]          ;Move to next location
```

Related Keyword

BREAK (program instruction)

Syntax

BREAK

Function

Suspend program execution until the current motion completes.

Usage Considerations

The BREAK instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the robot is not attached to the task executing the BREAK instruction, the instruction completes immediately.

If the V+ system is not configured to control a robot, executing the BREAK instruction will cause an error.

Details

This instruction has two effects:

1. Program execution is suspended until the robot reaches its current destination.

   Note, however, that BREAK cannot be used to have one task wait until a motion is completed by another task.

2. The continuous-path transition between the current motion and that commanded by the next motion instruction is broken. That is, the two motions are prevented from being merged into a single continuous path.

Related Keywords

BRAKE (program instruction)

CP (system switch)

SELECT (program instruction and real-valued function)

Syntax

BSTATUS

Function

Return information about the status of the conveyor tracking system.

Usage Considerations

This option is available only if your Adept system is equipped with the V+ Extensions option.

The BSTATUS function returns information for the robot selected by the task executing the function.

The word bstatus cannot be used as a program name or variable name.

Details

This function returns a value that is equivalent to the binary value represented by a set of bit flags, which indicate the following conditions of the conveyor tracking software:

Bit 1 (LSB)	Tracking belt (mask value = 1) When this bit is set, the robot is currently tracking a belt.
Bit 2	Destination upstream (mask value = 2) When this bit is set, the destination location was found to be upstream of the belt window during the planning of the last motion.
Bit 3	Destination downstream (mask value = 4) When this bit is set, the destination location was found to be downstream of the belt window during the planning of the last motion.
Bit 4	Window violation (mask value = 8) When this bit is set, a window violation occurred while the robot was tracking a belt during the last belt-relative motion. (This flag is cleared at the start of each belt-relative motion.)

Related Keywords

BELT	(system switch and real-valued function)
BELT.MODE	(system parameter)
DEFBELT	(program instruction)
SELECT	(program instruction and real-valued function)
WINDOW	(program instruction and real-valued function)

Syntax

```
... value BXOR value ...
```

Function

Perform the **binary** exclusive-OR operation on two values.

Usage Considerations

The BXOR operation is meaningful only when performed on integer values.

Details

The BXOR operator can be used to perform a binary exclusive-OR operation on two values on a bit-by-bit basis, resulting in a real value. Note, however, that this operation is meaningful only when performed on integer values.

Specifically, the BXOR operation consists of the following steps:

1. Convert the operands to sign-extended 24-bit integers, truncating any fractional part.
2. Perform a binary exclusive-OR operation (see below).
3. Convert the result back to a real value.

During the binary exclusive-OR operation,

```
c = a BXOR b
```

the bits in the resultant C are determined by comparing the corresponding bits in the operands A and B as indicated in **Table 2-5 on page 106**.

Table 2-5. Converting A and B Operands into Real Values (BXOR Operator)

For each bit in: a	b		c
0	0	➡	0
0	1	➡	1
1	0	➡	1
1	1	➡	0

That is, a bit in the result will be 1 if the corresponding bit in one (and only one) of the operands is 1.

Refer to the ***V+ Language User's Guide*** for the order in which operators are evaluated within expressions.

Examples

Consider the following (binary values are shown only to make the operation more evident):

^B101000 BXOR ^B100001 yields ^B001001 (9.0)

Note that a very different result is obtained with the logical XOR operation:

^B101000 XOR ^B100001 yields 0 (FALSE)

In this case, ^B101000 and ^B100001 are each interpreted as logically TRUE since they are nonzero.

Related Keywords

BAND (operator)

BOR (operator)

XOR (operator)

Syntax

SCALE(transformation BY *value*)

SHIFT(transformation BY *value, value, value*)

Function

Complete the syntax of the SCALE and SHIFT functions.

Examples

```
SET new.trans = SCALE(old.trans BY scale.factor)
SET new.trans = SHIFT(old.trans BY x,y,z)
```

Related Keywords

SCALE (transformation function)

SHIFT (transformation function)

Syntax

```
CALIBRATE mode, status
```

Function

Initialize the robot positioning system with the robot's current position.

Usage Considerations

Normally, the instruction is issued with `mode` equal to 0.

The instruction has no effect if the **DRY.RUN** system switch is enabled.

If the robot is to be moved under program control, the CALIBRATE instruction (or command) must be processed every time system power is turned on and the V+ system is booted from disk.

The robot cannot be moved under program control or with the manual control pendant until a CALIBRATE instruction (or monitor command) has been processed.[1]

If multiple robots are connected to the system controller, this instruction attempts to calibrate all robots unless they are disabled with the ROBOT switch. All of the enabled robots must be calibrated before any of them can be moved under program control.

The CALIBRATE instruction may operate differently for each type of robot. For Adept robots, this instruction causes the robot to move. The robot must be far enough from the limits of the working range that it will not move out of range during the calibration process. (See the description of the CALIBRATE monitor command for details of the robot motion.)

This instruction generates an error if the robot is not selected by the task executing the CALIBRATE instruction. (See the **SELECT** instruction.)

If the V+ system is not configured to control a robot, executing the CALIBRATE instruction causes an error.

[1] Some robots can be moved in joint mode with the MCP even when they have not been calibrated.

Parameters

`mode` An optional real expression that indicates what part of the calibration process is to be performed:

Value of mode	Interpretation
–1	Set the V+ system into the calibrate mode. This is a special robot control mode that ensures that no robot motions are generated that might conflict with those commanded by the user calibration program. This system mode is terminated when the robot is successfully calibrated (as indicated by a modification to the NOT.CALIBRATED system parameter) or when an error occurs such as the disabling of robot power.
0 (or omitted)	Optionally load the calibration program; run the calibration program (with load, execute, and delete flags set).
1	Optionally load the calibration program; run the calibration program (with load flag set).
2	Run the calibration program (with the execute flag set).
3	Run the calibration program (with the delete flag set).

`status` An optional real variable that receives the exit status returned by the user calibration program or (in mode –1) from the V+ system when trying to enter into the special calibrate mode. If it completed with success, `status` = 1; otherwise, `status` is a standard V+ error code.

Details

When started, V+ assumes that the robot is not calibrated and restricts your ability to move the robot—with the manual control pendant (MCP) or an application program. Note that the COMP mode light on the MCP does not come on when the robot is not calibrated.

The robot loses start-up calibration whenever system power is switched off. As a safety measure, Adept robots also lose start-up calibration whenever an *Encoder quadrature error* occurs for one of the robot joints. Other servo errors that can cause the robot to lose calibration are *Unexpected zero index*, *No zero index*, and *RSC Communications Failure*.

The robot must be selected by the program that executes the CALIBRATE instruction. The default selection is robot #1. You can use the SELECT ROBOT instruction to select another robot. If the robot is not attached by the program, the robot is temporarily attached while the CALIBRATE instruction is executed.

For Adept robots, this instruction causes all the robot joints to be driven.

For devices controlled by the AdeptMotion VME option, the robot action depends upon the particular robot device module that is being used.

When this program instruction attempts to load the calibration file, it uses the same file name and search algorithm that the CALIBRATE monitor command uses. This instruction runs the user calibration program in the task that executed the invoking CALIBRATE program instruction.

If you wish to carry out a CALIBRATE instruction in task 0, one way to do so is from a program run using the /C qualifier on the EXECUTE instruction. With that qualifier specified, a program to calibrate the robot can run in task 0 even when DRY.RUN is disabled. A program running in any task other than 0 can execute the CALIBRATE instruction without special conditions.

Example

The following instruction sequence can be used by any program task to perform start-up calibration on the robot (if task #0 is used, the DRY.RUN switch must be enabled before the program is executed):

```
DETACH()          ;Detach the robot
DISABLE DRY.RUN   ;Ensure DRY.RUN is disabled
CALIBRATE         ;Calibrate the robot
ATTACH()          ;Reattach the robot
```

Related Keywords

CALIBRATE	(monitor command, see the ***V+Operating System Reference Guide)***
NOT.CALIBRATED	(system parameter)
SELECT	(program instruction and real-valued function)

Syntax

CALL program(arg_list**)**

Function

Suspend execution of the current program and continue execution with a new program (that is, a subroutine).

Parameters

program Name of the new program to be executed.

arg_list Optional list of subroutine arguments (separated by commas) to be passed between the current program and the new program. (If no argument list is specified, the parentheses after the program parameter can be omitted.)

Arguments can be used to pass data to the called program, to receive results back, or a combination of both. (How arguments are passed is described below.)

Each argument can be any one of the data types supported by V+ (that is, belt, precision point, real-value, string, and transformation), and can be specified as a constant, a variable, or an expression.[1] The type of each argument must match the type of its counterpart in the argument list of the called program. An argument specified as a variable can be a simple variable, an array element, or an array with one or more of its indexes left blank. (See below for more information.)

Any argument can be omitted, with the result that the corresponding argument in the called program will be undefined. If an argument is omitted within the argument list, the separating comma must still be included. If an argument is omitted at the end of the list, the comma preceding the argument can also be omitted. (See the description of .PROGRAM for more information on the effect of omitting an argument.)

[1] If a value is being passed back to the calling program, the parameter must be specified as a variable.

Details

The CALL instruction causes execution of the current program to be suspended temporarily. Execution continues at the first step of the indicated new program, which is then considered a subroutine.

Execution automatically returns to the current program when a RETURN instruction is executed in the subroutine. Execution continues with the instruction immediately following the CALL instruction.

Subroutine arguments can be passed by value or by reference. When an argument is passed by value, a copy of the argument value is passed to the subroutine. Any changes to the corresponding subroutine argument in the subroutine will **not** be passed back to the calling routine. Any argument that is specified as an expression (or compound transformation) will be passed by value.

When an argument is a (scalar or array) variable, it is passed by reference. That means a **pointer** to the variable is passed to the subroutine, which then works with exactly the same variable as the calling routine. If the called routine changes the value of the variable, the value is also changed for the calling routine. This can be especially significant, for example, if the same variable is passed as two arguments of a subroutine call. Then, any change to either of the corresponding subroutine arguments in the subroutine would automatically change the other corresponding subroutine argument.

Note that an argument that would be passed by reference (because it is a variable) can generally be forced to passage by value. The way that is done depends on the type of the variable, as follows:

- For a real variable, passage by value can be forced by enclosing the variable in parentheses:

  ```
  CALL prog_a((count))
  ```

- For a string variable, an empty string () can be added to the variable:

  ```
  CALL prog_b($str.var+"")
  ```

- For a transformation variable (for example, start), an equivalent transformation value can be specified by a compound transformation consisting of the variable and the NULL transformation:

  ```
  CALL prog_c(start:NULL).
  ```

As stated above, the items in the `arg_list` must match their corresponding items in the called program. In addition to straightforward matches of scalar to scalar, and arrays of equal numbers of dimensions, there are several situations in which higher dimension arrays can be passed in place of lower dimension arrays. For example, all the following cases are valid:

Array element passed to a scalar:

```
CALL example(a[1])    ---> .PROGRAM example(b)
CALL example(a[1,2])  ---> .PROGRAM example(b)
CALL example(a[1,2,3])---> .PROGRAM example(b)
```

One dimension of an array passed to a one-dimensional array:

```
CALL example(a[])     ---> .PROGRAM example(b[])
CALL example(a[1,])   ---> .PROGRAM example(b[])
CALL example(a[1,2,]) ---> .PROGRAM example(b[])
```

Two dimensions of an array passed to a two-dimensional array:

```
CALL example(a[,])    ---> .PROGRAM example(b[,])
CALL example(a[1,,])  ---> .PROGRAM example(b[,])
```

Three dimensions of an array passed to a three-dimensional array:

```
CALL example(a[,,])   ---> .PROGRAM example(b[,,])
```

Examples

```
CALL pallet(count)
```

Branches to the program named pallet, passing to it a pointer to the variable count. When a RETURN instruction is executed, control returns to the program containing the CALL instruction and count will contain the current value of the corresponding subroutine argument.

```
CALL cycle(1, , n+3)
```

Branches to the program named cycle. The value 1 is passed to the first parameter of cycle, its second parameter will be undefined, and its third parameter will receive the value of the expression n+3. (If cycle has more than three parameters, the remainder will all be undefined.)

Since none of the arguments in the CALL are variables, no data will be returned by the program cycle.

Related Keywords

CALLS (program instruction)

.PROGRAM (program instruction)

RETURN (program instruction)

Syntax

```
CALLP var(arg_list)
```

Function

Call a program given a pointer to the program in memory.

Usage Considerations

Using SYMBOL.PTR and CALLP is an alternative to using CALLS to invoke a V+ subroutine given its name as string data. For some applications, the SYMBOL.PTR-CALLP combination is much faster than CALLS.

Parameters

`var`	A real variable (not an expression) that contains a pointer to a program symbol.
`arg_list`	A list of arguments to be passed between the current program and the new program.

Details

In situations where the same program is called multiple times, the CALLP instruction can be significantly more efficient than the CALLS instruction. This is especially true in systems that have many programs loaded. (In situations where a program is called only once, the CALLS instruction is faster.)

When a CALLS instruction is used, the following steps are performed *each time* the CALLS instruction is encountered:

1. The user string is evaluated.
2. The V+ program symbol with that name is found in the V+ symbol table.
3. The proper V+ program is called.

As an alternative, the SYMBOL.PTR function can be used to perform the first two steps. Typically, that is done one time, during the initialization portion of the application software. Then, in place of the CALLS instruction, a CALLP instruction can be used to perform the third step above. (The CALLP instruction is just slightly slower than a CALL instruction.)

In situations where the same program is called multiple times, avoiding the first two steps of CALLS can be significant, especially in systems that have many programs loaded.

The CALLP instruction calls the program pointed to by the real variable **var**. This variable should have been obtained by using the SYMBOL.PTR function. If the value of **var** is zero, no program is called, and no error is reported. If **var** does not contain a valid pointer, program execution stops with error –406 (*Undefined program or variable name*).

Example

Instructions such as the following are executed in the initialization section of the application program:

```
my.pgm.ptr[1] = SYMBOL.PTR("my.program.1")
my.pgm.ptr[2] = SYMBOL.PTR("my.program.2")
```

Then, in the repetitive section of the application program, the following is executed:

```
CALLP my.pgm.ptr[index](parm1, parm2)
```

Related Keywords

CALLS (program instruction)

$SYMBOL (string function)

SYMBOL.PTR (real-valued function)

Syntax

```
CALLS string(arg_list)
```

Function

Suspend execution of the current program and continue execution with a new program (that is, a subroutine) specified with a string value.

Usage Considerations

CALLS takes much longer to execute than the normal CALL instruction. Thus, CALLS should be used only when necessary.

Parameters

`string` String value, variable, or expression defining the (1- to 15-character) name of the new program to be executed. (The letters in the name can be lowercase or uppercase.)

`arg_list` Optional list of arguments to be passed between the current program and the new program. The parentheses can be omitted if no argument list is specified. (See the **CALL** instruction for further information on subroutine arguments.)

NOTE: Since the argument list is not specified as part of the **`string`** parameter, all the subroutines called by a *specific* CALLS instruction must have equivalent argument lists.

Details

The CALLS instruction behaves exactly like the CALL instruction. The only difference between the two instructions is the way the subroutine name is specified. CALL requires that the name be explicitly entered in the instruction step. CALLS permits the name to be specified by a string variable or expression, which can have its value defined when the program is executed. That allows the program to call different subroutines depending on the circumstances.

As with the CALL instruction, execution automatically returns to the current program when a RETURN instruction is executed in the subroutine. Execution continues with the instruction immediately following the CALLS instruction.

Examples

The program segment below demonstrates how the CALLS instruction can be used to branch to a subroutine whose name is determined when the program is executed.

First the program reads a set of four digital input lines (1001 to 1004) to determine which of sixteen different part types it is dealing with. The part type is considered to be a hexadecimal number, which is converted to the corresponding ASCII character. Once the character is defined, the appropriate subroutine (that is, part.0, part.1, ..., part.f) is called to process the part. (The part-type value is also used to select the portion of the two-dimensional array argument that is passed to the subroutine.)

```
type = BITS(1001,4)         ;Get part type from digital input
$type = $ENCODE(/H0, type) ;Convert to ASCII character
CALLS "part."+$type(arguments[type,], status)
```

This example could be made more robust by using the STATUS real-valued function to make sure the proposed subroutine exists before it is called. That way the program could avoid possible errors from undefined program names.

Related Keywords

CALL (program instruction)

.PROGRAM (program instruction)

RETURN (program instruction)

Syntax

CASE value *OF*

Function

Initiate processing of a CASE structure by defining the value of interest.

Usage Considerations

This instruction must be part of a complete CASE structure.

Parameter

value	Real value, variable, or expression that defines the value to be matched in the CASE structure to determine which instructions are executed.

Details

This is perhaps the most powerful structure available with V+. It provides a means for executing one group of instructions from among any number of groups. The complete syntax is as follows (the blank lines are not required):

```
CASE value OF

   VALUE value_1,...:
     group_of_steps

   VALUE value_2,...:
     group_of_steps
     .
     .
     .

   ANY
     group_of_steps

END
```

The three vertical dots indicate that any number of VALUE steps can be used to set off additional groups of instructions.

The ANY step and the group of steps following it are optional. There can be only one ANY step in a CASE structure, and it must mark the last group in the structure (as shown above).

Note that the ANY and END steps must be on lines by themselves as shown. (However, as with all instructions, those lines can have comments.)

The CASE structure is processed as follows:

1. The expression following the CASE keyword is evaluated.
2. All the VALUE steps are scanned until the first one is found that has the same value.
3. The group of instructions following that VALUE step is executed.
4. Execution continues at the first instruction after the END step.

If no VALUE step is found that contains the same value as that in the CASE instruction, and there is an ANY step in the structure, then the group of instructions following the ANY step will be executed.

If no VALUE match is found in the structure, and there is no ANY step, none of the instructions in the entire CASE structure will be executed.

Examples

The following example shows the basic function of a CASE statement:

```
CASE number OF
 VALUE 1:
   TYPE "one"
 VALUE 2:
   TYPE "two"
 ANY
   TYPE "Not one or two"
END
```

The following sample program asks the user to enter a test value. If the value is negative, the program exits after displaying a message. Otherwise, a CASE structure is used to classify the input value as a member of one of three groups. The groups are (1) even integers from zero to ten, (2) odd integers from one to nine, and (3) all other positive numbers.

```
PROMPT "Enter a value from 1 to 10: ", x

CASE x OF
  VALUE 0, 2, 4, 6, 8, 10:
     TYPE "The number", x, " is EVEN"

  VALUE 1, 3, 5, 7, 9:
     TYPE "The number", x, " is ODD"

  ANY
     TYPE x, " is not an integer from 0 to 10"
END
```

The following example shows a special use of the CASE structure to test Boolean conditions:

```
PROMPT "Enter a number", x

CASE TRUE OF
  VALUE (x > 0):
     TYPE "The number was greater than 0."
  VALUE (x == 0):
     TYPE "The number was equal to 0."
  VALUE (x < 0):
     TYPE "The number was less than 0."
END
```

Related Keywords

ANY (program instruction)

END (program instruction)

VALUE (program instruction)

Syntax

$CHR (value)

Function

Return a one-character string corresponding to a given ASCII value.

Parameter

value Real-valued expression defining the value to be translated into a character. The value must be in the range of 0 to 255 (decimal).

If the value is in the range 0 to 127 (decimal), the corresponding ASCII character will be returned.

Example

```
$CHR(65)
;Returns the character A, since its ASCII value is 65.
```

Related Keywords

ASC (real-valued function)

$FLTB (string function)

$INTB (string function)

Syntax

```
CLEAR.EVENT task, flag, processor
```

Function

Clear an event associated with the specified task.

Parameters

task	Real value, variable, or expression (interpreted as an integer) that specifies the task for which the event is to be cleared. The valid range is 0 to 6 or 0 to 27, inclusive.[1]
flag	Not used, defaults to 1.
processor	Optional real value, variable, or expression that specifies the V+ processor running the task to be signaled. The default value is 0 which indicates the local processor. The maximum value depends on the software and hardware configuration. (Only systems equipped with the V+ Extensions option can have multiple processors.)

Details

This instruction clears the event associated with the specified task.

The default event cleared is the input/output completion event for which the instruction WAIT.EVENT 1 waits. This event is also cleared by the execution (not the completion) of an input/output instruction.

Related Keywords

GET.EVENT	(real-valued function)
INT.EVENT	(program instruction)
SET.EVENT	(program instruction)
WAIT.EVENT	(program instruction)

[1] The basic system allows 7 tasks (0 - 6). The V+ Extensions option allows 28 tasks (0 - 27).

Syntax

CLOSE

CLOSEI

Function

Close the robot gripper.

Usage Considerations

CLOSE causes the hand to close during the next robot motion.

CLOSEI causes a BREAK in the current continuous-path motion and causes the hand to close immediately after the current motion completes.

The CLOSE instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

The CLOSEI instruction can be executed by any program task so long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing these instructions will cause an error.

Details

These instructions send a signal to the control valves for the pneumatic hand to close. If the CLOSE instruction is used, the signal will not be sent until the next robot motion begins.[1]

The CLOSEI instruction differs from CLOSE in the following ways:

- A BREAK occurs if a continuous-path robot motion is in progress.
- The signal is sent to the control valves at the conclusion of the current motion, or immediately if no motion is in progress.
- Robot motions are delayed for a brief time to allow the hand actuation to complete. The length of the delay (in seconds) is the current setting of the HAND.TIME system parameter.

[1] You can use the SPEC Utility program to set which digital signals control the pneumatic hand. See the *Instructions for Adept Utility Programs* for information on use of the program.

Examples

During the next robot motion, cause the pneumatic control valves to assume the closed state:

```
CLOSE
```

Cause the pneumatic control valves to assume the closed state as soon as the current motion stops:

```
CLOSEI
```

Related Keywords

HAND.TIME	(system parameter)
OPEN and **OPENI**	(program instructions)
RELAX and **RELAXI**	(program instructions)
SELECT	(program instruction and real-valued function)

Syntax

COARSE tolerance ***ALWAYS***

Function

Enable a low-precision feature of the robot hardware servo.

Usage Considerations

Only the next robot motion will be affected unless the ALWAYS parameter is specified.

If the `tolerance` parameter is specified, its value becomes the default for any subsequent COARSE instruction executed during the current execution cycle (regardless of whether ALWAYS is specified).

FINE 100 ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The COARSE instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the COARSE instruction will cause an error.

Parameters

`tolerance`	Optional real value, variable, or expression that specifies the percentage of the standard coarse tolerances that are to be used for each joint of the robot attached by the current execution task. See the Details section for default values.
`ALWAYS`	Optional qualifier that establishes COARSE as the default condition. That is, COARSE will remain in effect until disabled by a FINE instruction. If ALWAYS is not specified, the COARSE instruction will apply only to the next robot motion.

Details

This instruction enables a low-tolerance feature in the robot motion servo so that larger errors in the final positions of the robot joints are permitted at the ends of motions. This allows faster motion execution when high accuracy is not required.

If the `tolerance` parameter is specified, the new setting takes effect at the start of the next motion. Also, the value becomes the default for any subsequent COARSE instruction executed during the current execution cycle (regardless of whether ALWAYS is specified). (Changing the COARSE tolerance does not affect the FINE tolerance.)

If the `tolerance` parameter is omitted, the most recent setting (for the attached robot) is used. The default setting is restored to 100 percent when program execution begins, or a new execution cycle starts (assuming that the robot is attached to the program).

Examples

Enable the low-tolerance feature only for the next motion:

```
COARSE
```

Enable the low-tolerance feature for the next motion, with the tolerance settings changed to 150% of the standard tolerance for each joint (that is, a looser tolerance):

```
COARSE 150
```

Enable the low-tolerance feature until it is explicitly disabled:

```
COARSE ALWAYS
```

Related Keywords

CONFIG (real-valued function)

FINE (program instruction)

NONULL (program instruction)

NULL (program instruction)

SELECT (program instruction and real-valued function)

Syntax

```
...  COM  value ...
```

Function

Perform the binary complement operation on a value.

Usage Considerations

The word com cannot be used as a program name or variable name.

The COM operation is meaningful only when performed on an integer value.

Parameter

value Real-valued expression defining the value to be complemented.

Details

The COM operator performs the binary complement operation on a bit-by-bit basis, resulting in a real value.

Specifically, the COM operation consists of the following steps:

1. Convert the operand to a sign-extended 32-bit integer, truncating any fractional part.
2. Perform a binary complement operation.
3. Convert the result back to a real value.

See the *V+ Language User's Guide* for the order in which operators are evaluated within expressions.

Examples

For example:

```
COM 40    yields   -41
```

Note that a very different result is obtained with the **logical** complement operation (NOT):

```
NOT 40    yields    0.0    (FALSE)
```

In this case, 40 is interpreted as logically TRUE since it is nonzero.

Syntax

```
CONFIG (select)
```

Function

Return a value that provides information about the robot's geometric configuration, or the status of the motion servo-control features.

Usage Considerations

The CONFIG function returns information for the robot selected by the task executing the function.

If the V+ system is not configured to control a robot, use of the CONFIG function causes an error.

Parameter

`select`	Optional real value, variable, or expression (interpreted as an integer) that has a value from 0 to 11 and selects the category of the configuration information to be returned. (See below for details.)

Details

This function returns a value that is interpreted as a series of bit flags. The meaning of the bit flags depends on the `select` parameter.

When the `select` parameter is omitted, or has the value 0, 1, or 2, the CONFIG function returns a value that can be interpreted as bit flags indicating a geometric configuration of the robot. That is, each bit in the value represents one characteristic of a robot configuration.

The parameter values in this group determine which robot configuration is returned by the function:

Select	Configuration information returned
0	The robot's current (instantaneous) configuration. (The default value is 0.)
1	The configuration the robot will achieve at the completion of the current motion, or the current configuration if no motion is in progress (and the robot is attached). (Note: The result returned is not meaningful if the robot is not attached.)
2	The configuration the robot will achieve at the completion of the next motion (assuming that it will be a joint-interpolated [not straight-line] motion).

The interpretations of the bit flags returned by these selections are as follows:

Bit #	Bit Mask	Indication if bit SET
1	1	Robot has righty configuration
2	2	Robot has below configuration
3	4	Robot has flipped configuration
4-16		(None)

When the `select` parameter is 3, 4, or 5, the CONFIG function returns a value that can be interpreted as bit flags indicating the settings of several robot motion servo-control features. That is, each bit in the value represents the state of one motion servo-control feature.

The different parameter values in this group select which motion(s) will be affected by the features settings reported by the function, as follows:

Select	Configuration information returned
3	The permanent settings of the robot motion servo-control features. That is, the settings defined by instructions that specify the ALWAYS qualifier.
4	The temporary settings for the motion currently executing, or the last motion completed if no motion is in progress.
5	The temporary settings that will apply to the next motion performed.

The interpretations of the bit flags returned by selections 3, 4, and 5 are as follows:

Bit#	Bit mask	Indication if bit CLEAR	Bit SET
1	1	(None)	(None)
2	2	FINE asserted	COARSE asserted
3	4	NULL asserted	NONULL asserted
4	8	MULTIPLE asserted	SINGLE asserted
5	^H10	CPON asserted	CPOFF asserted
6	^H20	OVERLAP asserted	NOOVERLAP asserted
7-16		(None)	(None)

When the `select` parameter is 6, 7, or 8, the CONFIG function returns a value that represents the setting of the FINE tolerance.

Select	FINE tolerance returned
6	The permanent setting, as a percentage of the standard tolerance.
7	The setting used for the previous or current motion, as a percentage of the standard tolerance.
8	The setting to be used for the next motion, as a percentage of the standard tolerance.

When the `select` parameter is 9, 10, or 11, the CONFIG function returns a value that represents the setting of the COARSE tolerance.

Select	COARSE tolerance returned
9	The permanent setting, as a percentage of the standard tolerance.
10	The setting used for the previous or current motion, as a percentage of the standard tolerance.
11	The setting to be used for the next motion, as a percentage of the standard tolerance.

When the `select` parameter is 12, the available joint configuration options for the selected robot are returned as shown below.

Bit #	Bit Mask	Indication if bit set
1	1	Robot can have lefty or righty configuration.
2	2	Robot can have above or below configuration.
3	4	Robot can have flipped or noflip configuration.
18	^H20000	Robot supports the OVERLAP and NOOVERLAP instructions.
22	^H200000	Robot's last rotary joint can be limited to ±180° (single/multiple).

When the `select` parameter is 13, the type of robot motion is returned. The bit values returned are shown below.

Bit #	Definition
^H1	This bit is set if the joint is interpolated, otherwise it is cleared for straight-line motion.
^H2	This bit is set if the robot is in a SPIN motion.

Related Keywords

ABOVE (program instruction)

BELOW (program instruction)

COARSE (program instruction)

CPOFF (program instruction)

CPON (program instruction)

FINE (program instruction)

FLIP (program instruction)

LEFTY (program instruction)

MULTIPLE (program instruction)

NOFLIP (program instruction)

NONULL (program instruction)

NOOVERLAP (program instruction)

NULL (program instruction)

OVERLAP (program instruction)

RIGHTY (program instruction)

SELECT (program instruction and real-valued function)

SINGLE (program instruction)

STATE (real-valued function)

Syntax

COS (value)

Function

Return the trigonometric cosine of a given angle.

Usage Considerations

The angle parameter must be measured in degrees.

The parameter will be interpreted as modulo 360 degrees, but excessively large values may cause a loss of accuracy in the returned value.

Parameter

value	Real-valued expression that defines the angular value (in degrees) to be considered.

Details

Returns the trigonometric cosine of the argument, which is assumed to be in degrees. The resulting value will always be in the range of –1.0 to +1.0, inclusive.

Examples

```
COS(0.5)       ;Returns 0.999962
COS(-5.462)    ;Returns 0.9954596
COS(60)        ;Returns 0.4999999
COS(1.3125E+2) ;Returns -0.6593457
```

NOTE: TYPE, PROMPT, and similar instructions output the results of the above examples as single-precision values. However, they are actually stored and manipulated as double-precision values. The LISTR monitor command will display real values to full precision.

Syntax

```
... CP
```

Function

Control the continuous-path feature.

Details

The CP switch can be used to turn off continuous-path motion processing. Refer to the *V+ Language User's Guide* for an explanation of continuous-path motions.

This switch is **enabled** when the V+ system is initialized.

Example

```
DISABLE CP ;Turn off continuous-path motion processing.
```

Related Keywords

BREAK	(program instruction)
DISABLE	(monitor command and program instruction)
ENABLE	(monitor command and program instruction)
SWITCH	(monitor command, program instruction, and real-valued function)

Syntax

CPOFF ***ALWAYS***

Function

Instruct the V+ system to stop the robot at the completion of the next motion instruction (or all subsequent motion instructions) and null position errors.

Usage Considerations

Only the next robot motion will be affected if the ALWAYS parameter is not specified.

CPON ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The CPOFF instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies only to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the CPOFF instruction will cause an error.

Parameter

ALWAYS	Optional qualifier that establishes CPOFF as the default condition. That is, when ALWAYS is included in a CPOFF instruction, CPOFF will remain in effect continuously until disabled by a CPON instruction. If ALWAYS is not specified, the CPOFF instruction applies only to the next robot motion.

Details

When CPOFF is in effect, the robot will be brought to a stop at the completion of the next robot motion, and any final position errors will be nulled (if required).

Unlike the BREAK instruction, which is executed **after** a motion to cause continuous-path processing to terminate, CPON and CPOFF are executed **before** a motion instruction to affect the continuous-path processing of the next motion instruction. Also, while BREAK applies to only one motion instruction, CPOFF can apply to all the motion instructions that follow.

If the CP system switch is disabled, continuous-path processing never occurs regardless of any CPON or CPOFF instructions.

Related Keywords

BREAK (program instruction)

CP (system switch)

CPON (program instruction)

SELECT (program instruction and real-valued function)

Syntax

CPON ***ALWAYS***

Function

Instruct the V+ system to execute the next motion instruction (or all subsequent motion instructions) as part of a continuous path.

Usage Considerations

Only the next robot motion will be affected if the ALWAYS parameter is not specified.

This is the default state of the V+ system. CPON ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The CPON instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies only to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the CPON instruction causes an error.

Parameter

ALWAYS	Optional qualifier that establishes CPON as the default condition. That is, if ALWAYS is specified, CPON will remain in effect continuously until disabled by a CPOFF instruction. If ALWAYS is not specified, the CPON instruction applies only to the next robot motion.

Details

When CPON is in effect, it is possible to execute a series of motion instructions that are blended into a single continuous path. That is, each motion will be performed in succession without stopping the robot at specified locations.

Unlike the BREAK instruction, which is executed **after** a motion to cause continuous-path processing to terminate, CPON and CPOFF are executed **before** a motion instruction to affect the continuous-path processing of the next motion instruction.

While asserting CPON permits continuous-path processing to occur, any of the following conditions will break a continuous path and override CPON:

- No subsequent motion instruction is executed before completion of the next motion instruction.
- CP system switch is disabled.

  (If the CP system switch is disabled, continuous-path processing never occurs, regardless of any CPON or CPOFF instructions.)
- The next motion instruction is followed by an instruction that explicitly or implicitly causes motion termination (for example, BREAK, OPENI).

Related Keywords

BREAK (program instruction)

CP (system switch)

CPOFF (program instruction)

SELECT (program instruction and real-valued function)

Syntax

```
CYCLE.END task_num, stop_flag
```

Function

Terminate the executing program in the specified task the next time it executes a STOP program instruction (or its equivalent).

Suspend processing of an executable program until a program running in the specified task completes execution.

Usage Considerations

The CYCLE.END instruction has no effect if the specified program task is not active.

The CYCLE.END instruction suspends execution of the program containing the instruction until the specified program task completes execution. If a program is aborted while its execution is suspended by a CYCLE.END instruction, the program task specified by the CYCLE.END instruction will still be terminated (if the `stop_flag` is TRUE).

Parameters

`task_num` Optional real value, variable, or expression (interpreted as an integer) that specifies which program task is to be monitored or terminated.

If the task number is not specified, the CYCLE.END *instruction* always accesses task #0.

`stop_flag` Optional real value, variable, or expression interpreted as a logical (TRUE or FALSE) value. If the parameter is omitted or has the value 0, the specified task is *not* stopped—but the CYCLE.END has all its other effects (see below). If the parameter has a nonzero value, the selected task stops at the end of its current cycle.

Details

If the `stop_flag` parameter has a TRUE value, the specified program task will terminate the next time it executes a STOP program instruction (or its equivalent), regardless of how many program cycles are left to be executed.

> **NOTE:** CYCLE.END will not terminate a program with continuous *internal* loops. Such a program must be terminated with the ABORT command or instruction.

Regardless of the `stop_flag` parameter, this instruction will wait until the program actually is terminated. If the program being terminated loops internally so that the current execution cycle never ends, the CYCLE.END instruction will wait forever.

To proceed from a CYCLE.END that is waiting for a program to terminate, abort the program that is waiting for a CYCLE.END by typing an ABORT command for the program task that executed the CYCLE.END instruction.

Example

The following program segment shows how a program task can be initiated from another program task (the ABORT and CYCLE.END program instructions are used to make sure the specified program task is not already active):

```
ABORT 3                 ;Abort any program already active
CYCLE.END 3             ;Wait for execution to abort
EXECUTE 3 new.program ;Start up the new program
```

Related Keywords

ABORT (monitor command and program instruction)

CYCLE.END (monitor command, see the *V+ Operating System Reference Guide)*

EXECUTE (monitor command and program instruction)

KILL (monitor command and program instruction)

STATUS (monitor command and real-valued function)

STOP (program instruction)

Syntax

DBLB ($string, first_char**)**

Function

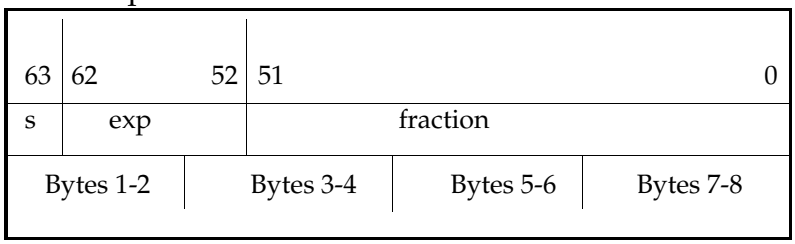
Return the value of eight bytes of a string interpreted as an IEEE double-precision floating-point number.

Parameters

$**string**	String expression that contains the eight bytes to be converted.
first_char	Optional real-valued expression that specifies the position of the first of the eight bytes in the string. If first_char is omitted or has a value of 0 or 1, the first eight bytes of the string are extracted. If first_char is greater than 1, it is interpreted as the character position for the first byte. For example, a value of 2 means that the second through ninth bytes are extracted. An error is generated if first_char specifies eight bytes that are beyond the end of the input string.

Details

Eight sequential bytes of the given string are interpreted as being a double-precision (64-bit) floating-point number in the IEEE standard format. This 64-bit field is interpreted as follows:



where

s is the sign bit, s = 0 for positive, s = 1 for negative.

exp is the binary exponent, biased by –1023.

fraction is a binary fraction with an implied 1 to the left of the binary point.

For 0 < exp < 2047, the value of a floating point number is:

$-1^{s} * (1.\text{fraction}) * 2^{\text{exp} - 1023}$

Double-precision real values have the following special values:

exp	fraction	Description
0	Zero	Zero value
0	Nonzero	Denormalized number
2047	Zero	Signed infinity
2047	Nonzero	Not-a-number

The range for normalized numbers is approximately 2.2^{-308} to 18^{307}

The main use of this function is to convert a binary floating-point number from an input data record to a value that can be used internally by V+.

Example

```
DBLB($CHR(^H3F)+$CHR(^HF0)+$CHR(0)+$CHR(0)
+$CHR(0)+$CHR(0)+$CHR(0)+$CHR(0)) ;Returns 1.0
```

Related Keywords

ASC (real-valued function)

$DBLB (string function)

FLTB (real-valued function)

$FLTB (string function)

INTB (real-valued function)

TRANSB (real-valued function)

VAL (real-valued function)

Syntax

```
$DBLB (value)
```

Function

Return an 8-byte string containing the binary representation of a real value in double-precision IEEE floating-point format.

Parameter

value	Real-valued expression, the value of which is converted to its IEEE floating-point binary representation.

Details

A real value is converted to its binary representation using the IEEE double-precision standard floating-point format. This 64-bit value is packed as eight successive 8-bit characters in a string. See the DBLB real-valued function for a more detailed description of IEEE floating-point format.

The main use of this function is to convert a double-precision real value to its binary representation in an output record of a data file.

Example

```
$DBLB(1.215)
```

Returns a character string equivalent to:

```
$CHR(^H3F)+$CHR(^H3F)+$CHR(^H70)+
$CHR(^HA3)+$CHR(^HD7)+$CHR(^H0A)+$CHR(^H3D)+$CHR(^H71)
```

Related Keywords

$CHR	(string function)
$FLTB	(string function)
FLTB	(real-valued function)
$INTB	(string function)
$TRANSB	(string function)

Syntax

```
DCB (value)
```

Function

Convert BCD digits into an equivalent integer value.

Usage Considerations

No more than four BCD digits can be converted.

The DCB function is most often used with the BITS real-valued function to decode input from the digital input signal lines.

Parameter

value Real value interpreted as a binary bit pattern representing up to four BCD digits.

NOTE: An error will be reported if any digit is not a valid BCD digit. That is, if a digit is greater than 9.

Example

If external input signals 1001-1008 (8 bits of input) receive two BCD digits from an external device, then the instruction

```
input = DCB(BITS(1001, 8))
```

sets the real variable input equal to the integer equivalent of the BCD input on the specified signals.

Related Keyword

BCD (real-valued function)

Syntax

```
... DECEL.100 [robot_num]
```

Function

Enable or disable the use of 100 percent as the maximum deceleration for the ACCEL program instruction.

Parameter

`robot_num`	Optional real value, variable, or expression (interpreted as an integer) that indicates the number of the robot affected.

Details

When DECEL.100 is enabled for the selected robot, the maximum deceleration percentage defined by the SPEC utility program is ignored, and a maximum deceleration of 100% is used instead. This maximum is used to limit the value specified by the ACCEL program instruction. By default, DECEL.100 is disabled for all robots.

Example

```
DECEL.100[2] ;Cause ACCEL to use 100% for maximum
             ;deceleration for robot #2
```

Related Keywords

ACCEL (program instruction and real-valued function)

SPEED (monitor command and program instruction)

Syntax

```
$DECODE ($string_var, string_exp, mode)
```

Function

Extract part of a string as delimited by given break characters.

Usage Considerations

$DECODE modifies the *input* variable as well as returning a string value.

The test for break characters is always performed without regard for the case of the characters in the input string.

The break characters are treated as individual characters, independent of the other characters in the string that defines them.

Parameters

`$string_var` String variable that contains the string to be scanned. After the function is processed, this variable will contain the portion of the original string that was *not* returned as the function value.

NOTE: This parameter is modified by the function and cannot be specified as a string constant or expression.

If the program causes the *same* variable to receive the function value, the variable will end up containing the value returned by the function.

`string_exp` String constant, variable, or expression that defines the individual break characters, which are to be considered as separating the substrings of interest in the input string value. (The order of the characters in this string has no effect on the function operation.)

`mode` Optional real value, variable, or expression that controls the operation performed by the function, as follows:

If the mode is negative or zero, or is omitted, characters up to the first break character are removed from the input string and returned as the output of the function.

If the mode is greater than zero, characters up to the first *non*break character are removed from the input string and returned as the output of the function. That is, this case returns all the leading break characters in the input string.

Details

This function is used to scan an input string and return the initial substring, as delimited by any of a group of break characters. After the substring is returned by the function, it is *deleted* from the input string.

The string returned (and deleted) can either contain no break characters (mode 0), or nothing but break characters (mode 1). That is, $DECODE can return (and delete) all the characters up to the first break character—usually some desired substring; or the function can return (and delete) all the leading break characters—which are usually discarded.

By alternating the modes, groups of desired characters can be picked from the input string (see the first example later).

The modes –2 and –3 copy all nonbreak characters up to the first break characters plus the first break character. Mode –2 is equivalent to the following instructions:

```
$s = $DECODE($i,$break,0);Extract up to 1st break character
$s = $s+$MID($i,1,1);Add on 1st break character
$i = $MID($i,2,127);Remove the break character
```

The following instruction can perform these operations:

```
$s = $DECODE($i,$break,-2);Extract through 1st break character
```

Mode –3 is equivalent to mode –2 if a break character is present. However, if no break character is contained in the input string, mode –3 returns an empty string and leaves the input string unchanged.

Examples

The instructions below pick off consecutive numbers from the string $input, assuming that the numbers are separated by some combination of spaces and commas.

The first instruction within the DO structure sets the variable $temp to the substring from $input that contains the first number (and removes that substring from $input). The VAL function is used to convert the numeric string into its corresponding real value, which is assigned to the next element of the real array value. The $DECODE function is used a second time to extract the characters that separate the numbers (the characters found are ignored).

```
i = 0                                ;Set array index
DO
   $temp = $DECODE($input," ,",0) ;Pick off a number string
   value[i] = VAL($temp)           ;Convert to real value
   $temp = $DECODE($input," ,",1) ;Discard spaces and
                                    ;commas
   i = i+1                          ;Advance the array index
UNTIL $input == ""                  ;Stop when input is empty
```

If $input contains a sequence of numeric values (as strings) separated by spaces, commas, or any combination of spaces and commas. That is, for example, the input string could have the value

```
$input = "1234. 93465.2, .4358,3458103"
```

Then the result of executing the above instructions will be that the first four elements of the value array will have the following values:

```
value[0] = 1234.0
value[1] = 93465.2
value[2] = 0.4358
value[3] = 3458103.0
```

Also, the string variable $input will end up containing an empty string ("").

As shown above, use of the $DECODE function normally involves two string variables: the input variable and the output variable. If you are interested only in the characters up to the first break character, and want to discard all the characters that follow, the same variable can be used for both input and output. In the following instruction, for example, the same variable is used on both sides of the equal sign because the programmer wants to discard all the white space (that is, space and tab) characters at the end of the input string. (Note that the break characters are specified by a string expression consisting of a space character and a tab character.)

```
$line = $DECODE($line," "+$CHR(9),0)
;Discard trailing blanks
```

Related Keywords

$TRUNCATE (string function)

$MID (string function)

Syntax

```
DECOMPOSE array_name[index] = location
```

Function

Extract the (real) values of individual components of a location value.

Parameters

array_name	Name of the real-valued array that will have its elements defined.
index	Optional integer value(s) that identifies the first array element to be defined. Zero will be assumed for any omitted index. If a multiple-dimension array is specified, only the right-most index will be incremented as the values are assigned.
location	Location value that will be decomposed into its component values. This can be a transformation value or a precision-point value, and can be defined by a variable or a location-valued function.

Details

This instruction assigns values to consecutive elements of the named array, corresponding to the components of the specified location.

If the location is represented as a transformation value, six elements are defined, corresponding to X, Y, Z, yaw, pitch, and roll.

If the location is represented as a precision-point value, then four, five, or six elements are defined (depending on the number of robot joints), that corresponds to the individual joint positions.

Examples

```
DECOMPOSE x[] = part
```

Assigns the components of transformation part to elements 0 to 5 of array x.

```
DECOMPOSE angles[4] = #pick
```

Assigns the components of precision point #pick to array element angles[4] and those that follow it.

Related Keywords

#PPOINT (precision-point function)

TRANS (transformation function)

Syntax

```
$DEFAULT ()
```

Function

Return a string containing the current system default device, unit, and directory path for disk file access.

Usage Considerations

Parentheses must be included even though the function has no parameters.

Details

The system default device, unit, and directory path can be set by the DEFAULT monitor command. The $DEFAULT function returns the current default values as a string. The string contains the portions of the following information that have been set:

```
device>disk_unit:directory_path
```

where:

`device`	is DISK, indicating a local disk drive, or KERMIT, indicating a serial line using the Kermit protocol.
`disk_unit`	is the disk unit specified to the DEFAULT monitor command. The colon (:) is omitted if no unit was specified.
`directory_path`	is any input to the DEFAULT command that followed the device and unit. The directory path is omitted if no additional input was specified.

Example

The following commands set the default disk specification to DISK>A:\TEST\, and then display it on the terminal for confirmation:

```
DEFAULT = D>A:\TEST\
LISTS $DEFAULT()
```

Related Keyword

DEFAULT (monitor command, see the ***V+ Operating System Reference Guide)***

Syntax

```
DEFBELT %belt_var = nom_trans, belt_num, vel_avg, scale_fact
```

Function

Define a belt variable for use with a conveyor tracking robot.

Usage Considerations

This option is available only if your Adept system is equipped with the V+ Extensions option.

The DEFBELT instruction supports up to six belt encoders, depending on the hardware configuration.

The BELT switch must be enabled for this instruction to be executed.

DEFBELT cannot be executed while the robot is moving relative to the specified belt variable.

When a belt variable is initialized using this instruction, its window parameters are set to allow any location in the working volume of the robot. That is, no belt window is set. (See the **WINDOW** instruction.)

When a belt variable is initialized with DEFBELT, error checking is initiated for the associated belt encoder. This error checking can be turned off by disabling the BELT system switch or by using the ZERO command to reinitialize the V+ system.

Parameters

%belt_var Name of the belt variable to be defined. (All appearances of belt variables must be prefixed with the percent character [%].)

nom_trans Transformation value that defines the position and orientation of the conveyor belt. This can be provided by a transformation variable, a transformation-valued function, or a compound transformation.

The X axis of the nominal transformation defines the direction of travel of the belt. Normally, the belt moves along the direction of +X. The X-Y plane defined by this transformation is parallel to the surface of the belt. The (X, Y, Z) position defined by the nominal transformation defines the approximate center of the belt with respect to the robot.

belt_num The number of the encoder used for reading the instantaneous location of the belt. (The V+ system can read up to two belt

encoders, numbered 1 and 2.) This can be specified with a constant, a variable, or an expression.

vel_avg (This parameter is currently ignored, but some value must be provided.)

scale_fact The calibration constant that relates motion of the conveyor belt with counts of the encoder mounted on the conveyor. This value (which can be supplied as a constant, a real variable, or an expression) will be interpreted as having the units in millimeters of belt travel per encoder count.

Details

A conveyor belt is modeled by a belt variable. In addition to the parameters to the DEFBELT instruction, a belt variable contains the following information:

- Window parameters, which define the working range of the robot along the belt. (Set with the WINDOW instruction.)
- An encoder offset, which is used to adjust the origin of the belt frame of reference. (Set with the SETBELT instruction.)

Belt variables have the following characteristics:

- Belt variable names must always be preceded by the percent character (%), for example, %main.belt. Otherwise, the normal rules for variable names apply.
- Belt variable arrays are allowed, for example, %b[x].
- Belt variables can be passed as subroutine parameters just like other variables.
- Belt variables can be defined only with the DEFBELT instruction—there is no assignment instruction for them. Thus, the following are *not* valid instructions:

```
%new_belt = %old_belt

SET %new_belt = %old_belt
```

- Belt variables cannot be stored on a mass-storage device. (Variables used to define the parameters in a DEFBELT instruction can be stored, however.)

Example

The following instruction will define the belt variable %belt.var. The value of b.num must be the number of the encoder to be associated with this belt variable (1 or 2). The variable b.num is also used as an index for arrays of data describing the position and orientation of the belt, its velocity smoothing, and the encoder scale factor.

```
DEFBELT %belt.var = belt.nom[b.num], b.num, v.avg[b.num],
belt.sf[b.num]
```

Related Keywords

BELT	(system switch and real-valued function)
BELT.MODE	(system parameter)
BSTATUS	(real-valued function)
SETBELT	(program instruction)
WINDOW	(program instruction and real-valued function)

Syntax

DEF.DIO signal = address, type

Function

Assign third-party digital I/O boards to standard V+ signal numbers, for use by standard V+ instructions, functions, and monitor commands.

Usage Considerations

This instruction requires the Third-Party Board Support license.

Parameters

signal An integer or real-valued expression representing a V+ signal number from 17 to 505 for output or from 1017 to 1505 for input. This is the first signal number of a block of 8 contiguous signals that is located in a byte at an address on the VMEbus. The signal numbers must be on an 8-bit boundary (17, 25, 33,...). The least-significant bit in the byte at the address corresponds to the lowest signal number.

address An integer or real-valued expression representing the VMEbus address in the ranges described in Details. (These are the same ranges as allowed for the IOGET_ and IOPUT_ keywords.)

type An optional integer or real-valued expression representing the VMEbus address type, like that specified for the IOGET_ and IOPUT_ keywords:

1 VMEbus standard address space (default)

2 VMEbus short address space

Details

The VMEbus specification defines three independent address spaces: short, standard, and extended. Adept does not support the extended address space. The DEF.DIO instruction can access addresses in the following ranges:

Standard	0 to ^H3FFFFF (0 to ^H7FFFFF on nonvision systems). With a dual vision system, no user memory is available in the standard space.
Short	0 to ^H7FFF

CAUTION: Adept must first approve all third-party boards used in Adept controllers. Contact Adept Applications support for details. Not all boards support the full range of addresses listed above.

After a DEF.DIO instruction has executed successfully, any operation that references any of the eight signals defined by the instruction will access the third-party board.

Example

Define input signals 1033 through 1040 to be associated with the third-party board that is configured to respond to the address ^H3000 in the short address space:

```
DEF.DIO 1033 = ^H3000, 2
```

Related Keywords

BITS	(real-valued function)
IO	(monitor command, see the *V+ Operating System Reference Guide)*
IOGET_	(real-valued functions)
$IOGETS	(string function)
IOPUT_	(program instructions)
RESET	(program instruction)
RUNSIG	(program instruction)
SIG	(real-valued function)
SIG.INS	(real-valued function)
SIGNAL	(program instruction)

Syntax

```
DEFINED (var_name)
```

Function

Determine whether a variable has been defined.

Parameter

var_name The name of a location, string, or real variable. Both scalar variables and array variables are permitted. A location variable can be a transformation, a precision point, or a belt variable.

Details

The value of the specified variable is tested. If the value is defined, the function returns the value TRUE. Otherwise, the value FALSE is returned.

For array variables, if a specific array element is specified, the single array element will be tested. If no array element is specified, this function will return a TRUE value if any element of the array is defined.

> **NOTE:** For nonreal arguments (i.e. strings, locations, transformations) that are passed in the argument list of a CALL statement, you can test to see if the variable is defined or not. However, you cannot assign a value to undefined nonreal arguments within the CALLed program. If you attempt to assign a value to an undefined nonreal argument, you will receive an undefined value error message.

Therefore, when using DEFINED to test for user input, be sure to assign a default value to the variable before testing it. See the example below.

Examples

```
.PROGRAM test($s)
AUTO $tmp
$tmp = "default"
IF DEFINED($s) THEN
   $tmp = $s
END
TYPE /C3 "The string is: ", $tmp
```

When the example above is executed with a value:

```
ex test("ABCD")
```

the routine returns:

```
The string is: ABCD
```

When the example above is executed without a value:

```
ex test()
```

the routine returns:

```
The string is: default
```

The instruction:

```
DEFINED(base_part)
```

returns a value of TRUE if the variable base_part is defined.

The instruction:

```
DEFINED(corner[])
```

returns a value of TRUE if any element of the array corner has been defined.

Related Keywords

STATUS	(real-valued function)
TESTP	(monitor command, see the ***V+ Operating System Reference Guide)***

Syntax

```
DELAY time
```

Function

Cause robot motion to stop for the specified time.

Usage Considerations

The robot will stop during the delay, but the wait and nulling normally associated with a motion BREAK do not occur.

Program execution will continue during the delay, up to the next motion instruction in the program. (V^+ system timers can be used to control the timing of program execution. The DELAY instruction should not be used for that purpose.)

The DELAY instruction is interpreted as a move-to-here motion instruction. (See below for the consequences of that interpretation.)

The DELAY instruction can be executed by any program task so long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the V^+ system is not configured to control a robot, executing the DELAY instruction will cause an error.

If the AMOVE instruction has been executed to prepare for motion of an extra axis, execution of the DELAY instruction will cancel the effect of the AMOVE instruction.

Parameter

`time` Real value, variable, or expression that specifies the length of time, in seconds, that the robot is to pause.

A time value less than 0.016 (16 milliseconds) will result in a 16-millisecond delay.

Details

The DELAY instruction is processed as a robot motion. As a result, the following consequences occur when a DELAY is executed:

1. Any pending hand actuation takes place during the execution of the DELAY instruction.
2. Any temporary trajectory switches that have been specified are cleared after the conclusion of the delay.
3. Any pending configuration change is canceled.

NOTE: When DRY.RUN mode is in effect, DELAY instructions will not cause any delay.

Examples

```
DELAY 2.5
```

Causes all robot motion to stop for 2.5 seconds and any pending hand operation to occur. Clears any temporary trajectory switches that may be set, and cancels any pending requests for configuration change.

```
DELAY pause.1
```

Stops all robot motion for pause.1 seconds.

Related Keywords

DURATION (program instruction)

SELECT (program instruction and real-valued function)

Syntax

```
DEPART distance

DEPARTS distance
```

Function

Start a robot motion away from the current location.

Usage Considerations

DEPART causes a joint-interpolated motion.

DEPARTS causes a straight-line motion, during which no changes in configuration are permitted.

These instructions can be executed by any program task so long as the task has attached a robot. The instructions apply to the robot selected by the task.

If the V+ system is not configured to control a robot, executing these instructions will cause an error.

Parameter

`distance` Real-valued expression that specifies the distance (in millimeters) along the robot tool Z axis between the current robot location and the desired destination.

A positive distance moves the tool back (toward negative tool-Z) from the current location; a negative distance moves the tool forward (toward positive tool Z).

Details

These instructions initiate a robot motion to a new location, which is offset from the current location by the distance given, measured along the current tool Z axis.

Examples

```
DEPART 80
```

Moves the robot tool 80 millimeters back from its current location using a joint-interpolated motion.

```
DEPARTS 2*offset
```

Withdraws the robot tool (2 * offset) millimeters along a straight-line path from its current location.

Related Keywords

APPRO and **APPROS** (program instructions)

MOVEF and **MOVESF** (program instructions)

SELECT (program instruction and real-valued function)

Syntax

DEST

Function

Return a transformation value representing the planned destination location for the current robot motion.

Usage Considerations

The DEST function returns information for the robot selected by the task executing the function.

If the V+ system is not configured to control a robot, use of the DEST function will not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

The word dest cannot be used as a program name or variable name.

Details

DEST returns the location a robot was moving to when its motion was interrupted. This applies for all motion instructions, including:

1. Motions to named locations, such as

   ```
   MOVE start
   ```

   ```
   MOVES #part[10], 25.40
   ```

   Note that even though the second instruction references a precision-point location variable, the DEST function would return a transformation value during that motion.

2. Motions to locations defined relative to named locations or defined relative to the current robot location

   ```
   APPROS drop, 50.00
   ```

   ```
   DEPART 30.00
   ```

   ```
   MOVE SHIFT(HERE BY 50,0,10)
   ```

3. Motions to special locations such as

   ```
   READY
   ```

The location value returned by the DEST function may not be the same as the location at which the robot stops if the motion of the robot is interrupted for some reason. For example, if the RUN/HOLD button on the manual control pendant is pressed, the robot will stop immediately, but DEST will still return the location the robot was moving to.

If a motion is not begun because V+ realizes the destination location cannot be reached (for example, it is too far from the robot), then DEST will **not** be set to the goal location.

Example

The DEST function is useful, for example, for continuing a motion that has been interrupted by a reaction initiated by a REACTI instruction. The subroutine automatically invoked could contain steps such as the following to process the interruption and resume the original motion.

```
SET save = HERE      ;Record where the robot is
SET old.dest = DEST ;Record where the robot was going
old.speed = SPEED(3);Record the current motion speed
DEPART 50.0          ;Back away a safe distance
   .
   .
   .
APPRO save, 50.0     ;Return to the original motion path
MOVES save           ;...back to where we left
SPEED old.speed      ;Restore the original motion speed
MOVES old.dest       ;Continue toward original destination
```

Related Keywords

HERE (transformation function)

#PDEST (precision-point function)

SELECT (program instruction and real-valued function)

Syntax

DETACH *(*logical_unit*)*

Function

Release a specified device from the control of the application program.

Usage Considerations

Detaching the robot causes a BREAK in continuous-path motion.

DETACH automatically forces an FCLOSE if a disk file or graphics window is open on the specified device.

The robot is automatically attached to task 0 when the EXECUTE monitor command or program instruction is processed to initiate that task and the DRY.RUN system switch is disabled. All the other logical units are automatically detached when program execution begins. (Other events that cause automatic detachment are listed below.)

Parameter

`logical_unit` Optional real value, variable, or expression (interpreted as an integer) that identifies the device to be detached. (See the ATTACH instruction for a description of logical unit numbers.)

The parentheses can be omitted if the logical unit number is omitted (causing the robot to be detached).

Details

This instruction releases the specified device from control by the application program. (No error is generated if the device was not attached.)

Control of the specified device can be returned to the program with the ATTACH instruction.

When `logical_unit` is 0 (or is omitted), the program releases control of the robot. While the robot is detached, robot power can be turned off and on, the manual control pendant can be used to move the robot, and a different robot can be selected (if more than one robot is connected to the system controller). A delay of one system cycle (16 ms) occurs when a robot is detached.

This is useful for applications that require the operator to define where the robot should be located for certain operations. For such tasks a teaching program can DETACH the robot and then output directions to the operator on the system terminal or the manual control pendant. Then the operator can use the manual control pendant to move the robot to the desired locations. The system terminal or the manual control can be used for accepting input from the operator (the latter can be read by using the PENDANT function).

When a disk logical unit is detached, any device that was specified by the corresponding ATTACH instruction is forgotten. Thus, a subsequent ATTACH instruction will need to specify the device again if the default device is not desired.

The following events automatically DETACH all the logical units (except the robot) from the affected program task:

- Processing of the EXECUTE command and instruction
- Processing of the KILL command and instruction
- Processing of the ZERO command
- Normal completion of program execution

  Note, however, that if a program terminates execution **abnormally**,[1] all of its devices remain attached—except that the terminal and the manual control pendant are detached. If the task is subsequently resumed, the program automatically reattaches the terminal and manual control pendant if they were attached before the termination.

  NOTE: It is possible that another program task could have attached the terminal or manual control pendant in the meantime. That would result in an error message when the stopped task is restarted.

```
DETACH      ;Release program control of the robot.

DETACH (1)  ;Discontinue program control of the
            ;manual control pendant.
```

Related Keyword

ATTACH (program instruction)

[1] Abnormal termination of program execution refers to any cause other than HALT or STOP instructions.

Syntax

```
     DEVICE (type, unit, error, p1, p2, ...) out[i], in[j],
out_trans, in_trans
```

Function

Send a command or data to an external device and, optionally, return data back to the program. (The actual operation performed depends on the device referenced.)

Usage Considerations

The syntax contains optional parameters that apply only to specific device types and commands.

Parameters

type	Real value, variable, or expression (interpreted as an integer) that indicates the type of device being referenced.
unit	Real value, variable, or expression (interpreted as an integer) that indicates the device unit number. The value must be in the range 0 to (max –1), where max is the maximum number of devices of the specified type. The value should be 0 if there is only one device of the given type.
error	Optional real **variable** that receives a standard system error number that indicates if this instruction succeeded or failed. If this parameter is omitted, any device error stops program execution. If an error variable is specified, the program must explicitly check it to detect errors.
p1, p2, ...	Optional real values, variables, or expressions, the values of which are sent to the device as part of a command. The number of parameters specified, and their meanings, depend upon the particular device type being accessed.
out[]	Optional real array that contains data values that are sent to the device as part of a command. The actual data sent depends upon the device type and command.
i	Optional real value, variable, or expression (interpreted as an integer) that indicates the first array element to be considered in the array out[]. Element 0 is accessed first if no index is specified.

`in[]`	Optional real array that receives any data values returned from the device as the result of a command. The actual data returned depends upon the device type and the command.
`j`	Optional real value, variable, or expression (interpreted as an integer) that indicates the first array element to be filled in the array `in[]`. Element 0 is accessed first if no index is specified.
`out_trans`	Optional transformation variable, function, or compound transformation that defines a transformation value to be sent to the device as part of a command.
`in_trans`	Optional transformation variable that receives a data value returned from the device as the result of a command. The actual data returned depends upon the device type and the command.

Details

DEVICE is a general-purpose instruction for accessing external devices. For details and examples see the ***V+ Language User's Guide*** for specific devices.

Related Keywords

DEVICE	(real-valued function)
DEVICES	(program instruction)
SETDEVICE	(program instruction)

Syntax

```
DEVICE (type, unit, error, p1, p2,...)
```

Function

Return a real value from a specified device. The value may be data or status information, depending upon the device and the parameters.

Usage Considerations

The syntax contains optional parameters that may be useful only for specific device types and commands.

Parameters

type	Real value that indicates the type of device being referenced.
unit	Real value that indicates the device unit number. The value must be in the range 0 to (max –1), where max is the maximum number of devices of the specified type. The value should be 0 if there is only one device of the given type.
error	Optional real variable that receives a standard system error number, which indicates if this function succeeded or failed. If this parameter is omitted, any device error stops program execution. If error is specified, the program must check it to detect errors.
p1, p2, ...	Optional real values that are sent to the device as part of a command. The number of values specified and the meanings of the values depend upon the particular device type.

Details

DEVICE is a general-purpose function for returning data and status information from external devices. For details and examples see the supplementary documentation for specific devices.

See the ***V+ Language User's Guide*** for information on use of the DEVICE function to access external encoders.

The DEVICE instruction is used to configure vision system memory allocation and frame buffer configuration. See the ***AdeptVision User's Guide*** for details.

Related Keywords

DEVICE (real-valued function)

DEVICES (program instruction)

SETDEVICE (program instruction)

Syntax

```
DEVICES (type, unit, error, p1, p2, ...) $out, $in
```

Function

Send commands or data to an external device and optionally return data. The actual operation performed depends on the device referenced.

Usage Considerations

The syntax contains optional parameters that may be useful only for specific device types and commands.

Parameters

type	Real value that indicates the type of device being referenced.
unit	Real value that indicates the device unit number. The value must be in the range 0 to (max –1), where max is the maximum number of devices of the specified type. The value should be 0 if there is only one device of the given type.
error	Optional real variable that receives a standard system error number that indicates if this instruction succeeded or failed. If this parameter is omitted, any device error stops program execution. If error is specified, the program must check it to detect errors.
p1, p2, ...	Optional real values that are sent to the device as part of a command. The number of values specified and the meanings of the values depend upon the particular device type.
$out	Optional string expression that defines a string to be sent to the device as part of a command. The actual data sent depends upon the device type and the command.
$in	Optional string variable that receives any data values returned from a device as the result of a command. The actual data returned depends upon the device type and the command.

Details

DEVICES is a general-purpose instruction for accessing external devices. It is similar to the DEVICE program instruction except that data items may be sent or received with strings rather than real arrays.

> **NOTE:** Similar to the CALL and CALLS instruction pair, this instruction is a string-based version of the DEVICE command. Thus, the name DEVICES can be thought of as device s, rather than devices.

For details and examples see the supplementary documentation for specific devices.

Related Keywords

DEVICE (real-valued function)

SETDEVICE (program instruction)

Syntax

```
DISABLE switch, ..., switch
```

Function

Turn off one or more system control switches.

Usage Considerations

If a specified switch accepts an index qualifier and the index is zero or omitted (with or without the brackets), *all* the elements of the switch array are enabled.

Parameter

`switch` Name of a system switch to be turned off.

The name can be abbreviated to the minimum length that uniquely identifies the switch. That is, for example, the MESSAGES switch can be referred to with ME since there is no other switch with a name beginning with the letters ME.

Details

System switches control various aspects of the operation of the V+ system, including some optional subsystems such as vision. The Switch entry in the index for this document directs you to the detailed descriptions of these switches.

Other system switches are available when options are installed. Refer to the option documentation for details. For example, the switches associated with the AdeptVision options are described in the ***AdeptVision Reference Guide***.

When a switch is disabled, or turned off, the feature it controls is no longer functional or available for use. Turning a switch on with the ENABLE monitor command or program instruction makes the associated feature functional or available for use.

NOTE: The system switches are shared by all the program tasks. Thus, care should be exercised when multiple tasks are disabling and enabling switches—otherwise, the switches may not be set correctly for one or more of the tasks. Disabling the DRY.RUN switch does not have effect until the next EXECUTE command or instruction is processed for task #0, an ATTACH instruction is executed for the robot, or a CALIBRATE command or instruction is processed.

The SWITCH monitor command or the SWITCH real-valued function can be used to determine the status of a switch at any time. The SWITCH program instruction can be used, like the DISABLE instruction, to disable a switch.

Example

```
DISABLE MESSAGES ;Turns off the MESSAGES switch.
```

Related Keywords

ENABLE (monitor command and program instruction)

SWITCH (monitor command, program instruction, and real-valued function)

Syntax

```
DISTANCE (location, location)
```

Function

Determine the distance between the points defined by two location values.

Parameter

`location` Transformation value that defines one of the points of interest. This can be a function, a variable, or a compound transformation.

Details

Returns the distance in millimeters between the points defined by the two specified locations. The order in which the locations are specified does not matter. Also, the orientations defined by the locations have no effect on the value returned.

Example

The statement

```
x = DISTANCE(HERE, part)
```

will set the value of the real variable x to be the distance between where the robot tool point is currently located to the point defined by the transformation part.

Related Keyword

IDENTICAL (real-valued function)

Syntax

DO

Function

Introduce a DO program structure.

Usage Considerations

The DO program structure must be concluded with an UNTIL instruction.

Details

The DO structure provides a way to control the execution of a group of instructions based on a control expression. The syntax for the DO structure is as follows:

```
DO
   group_of_steps
UNTIL logical_expression
```

Processing of the DO structure can be described as follows:

1. The group of instruction steps is executed.
2. The logical expression is evaluated. If the result is FALSE, return to item 1. Otherwise, proceed to item 3.
3. Program execution continues at the first instruction after the UNTIL step.

When this structure is used, it is assumed that some action occurs within the group of enclosed instructions that will change the result of the logical expression from TRUE to FALSE when the structure should be exited. Alternately, logical_expression could be replaced with an expression that evaluates the state of a digital I/O signal (see example).

Note that the group of instructions within the DO structure is **always** executed at least one time. (The WHILE structure differs in that respect.)

There do not need to be any instructions between the DO and UNTIL instructions. When there are no such instructions, the UNTIL criterion is continuously evaluated until it is satisfied, at which time program execution continues with the instructions following the UNTIL instruction.

Example

The following example uses a DO structure to control a task that involves moving parts from one place to another. The sequence assumes that the digital signal line buffer.full changes to the on state when the parts buffer becomes full. (The robot should then perform a different sequence of motions.)

```
.
.
DO
   CALL get.part()
   CALL put.part()
UNTIL SIG(buffer.full)
.
.
```

Related Keywords

DO	(monitor command, see the *V+ Operating System Reference Guide)*
DOS	(program instruction)
EXIT	(program instruction)
NEXT	(program instruction)
UNTIL	(program instruction)
WHILE	(program instruction)

Syntax

```
DOS string, error
```

Function

Execute a program instruction defined by a string expression.

Usage Considerations

Before the instruction is executed, the string must be translated from ASCII into the internal representation used by V+. Thus, the instruction *executes much more slowly* than a normal program instruction.

The string cannot define a declaration statement or most of the control structure statements.

The DOS instruction will be ignored if the string defines a comment line or a blank line.

Parameters

`string`	String constant, variable, or expression that defines the program instruction to be executed. The instruction may contain a label field (which is ignored) and may be followed by a standard comment field. Leading and trailing spaces and tabs are ignored.
`error`	Optional real variable that receives any parsing or execution error generated by the instruction. The value is set to 1 if the instruction succeeds. If the instruction fails, a standard V+ error number is returned. If this parameter is omitted and an error occurs, execution of the program stops and the appropriate error message is displayed.

Details

The DOS (DO String) instruction provides a means for modifying a program on the fly. That is, the embedded program instruction, which is defined by a string expression, is executed as though it had been entered in the program as a normal instruction.

The instruction executes in the context of the current program. Thus, any subroutine arguments, automatic variable, or local variable can be accessed.

If a variable referenced in the instruction is not found in the current program context, the variable is assumed to be global. Any new variables that are created by the instruction (for example, in an assignment statement) are created as globals. Normal variable type checking is performed, and errors are generated if there are type conflicts.

The single-line control statements GOTO, IF ... GOTO, CALL, and CALLS are allowed and execute normally. The multiline control structures (for example, CASE ... END, IF ... ELSE ... END) cannot be executed by the DOS instruction.

Examples

```
DOS "var = 123"
```

Causes the variable var to be assigned the value 123. If var is undefined, a new *global* variable named var is created. Any errors cause the program to stop executing.

```
DOS $ins, status
```

Causes the instruction contained in the string variable $ins to be executed. If an error occurs, a V+ error code is placed in the real variable status and execution continues.

Related Keywords

DO	(monitor command, see the ***V+ Operating System Reference Guide)***
MCS	(program instruction)

Syntax

```
DRIVE joint, change, speed
```

Function

Move an individual joint of the robot.

Usage Considerations

The DRIVE instruction can be executed by any program task so long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the DRIVE instruction causes an error.

If the AMOVE instruction has been executed to prepare for motion of an extra axis, execution of the DRIVE instruction cancels the effect of the AMOVE instruction.

Parameters

joint Number of the robot joint to be moved. This can be specified by a constant, a variable, or an expression.

change The change desired in the joint position. This can be specified by a constant, a variable, or an expression. The value can be positive or negative.

The value is interpreted in the units used to measure the joint position. That is, a change for a rotary joint must be the number of degrees the joint is to move; a change for a linear joint must specify the number of millimeters to move.

speed The temporary program speed to be used for the motion, considered as a percentage of the current monitor speed setting. Again, this can be specified by a constant, a variable, or an expression.

Details

Operates the single specified robot joint, changing its position by **change** amount (in degrees or millimeters). The joint number, **joint**, can be 1, 2, ..., n, where n is the number of joints the robot has.

The speed of the motion is governed by a combination of the speed given in this instruction and the monitor SPEED setting. That is, the regular program speed setting is not used. (See the SPEED monitor command and the SPEED program instruction for explanations of motion speeds.)

The duration setting established by the DURATION instruction also affects the execution time of the motion.

Example

Change the angle of joint 2 by driving the joint 62.4 degrees in the negative direction at a speed of 75% of the monitor speed:

```
DRIVE 2,-62.4,75
```

Related Keyword

SELECT (program instruction and real-valued function)

Syntax

```
... DRY.RUN
```

Function

Control whether or not V+ communicates with the robot.

Usage Considerations

The DRY.RUN switch can be enabled or disabled by an application program, but the new setting of the switch will not take effect until the next time any of the following events occurs:

1. An EXECUTE command or instruction is processed for task #0
2. The robot is attached with an ATTACH instruction
3. A CALIBRATE command or instruction is processed

Before an application program changes the setting of the DRY.RUN switch, the program must have the robot detached. Otherwise, an error will result when the attempt is made to change the switch setting.

Details

This system switch can be used to stop V+ from sending motion commands to the robot and expecting position information back from the robot. Thus, when the system is in DRY.RUN mode, application programs can be executed to test for such things as proper logical flow and correct external communication without having to worry about the robot running into something. (Also see the TRACE system switch.)

The manual control pendant can still be used to control the robot while the system is in DRY.RUN mode.

The DRY.RUN switch is sampled whenever a robot is attached. (Note that task #0 attempts to attach the robot when program execution begins or is resumed.) The DRY.RUN setting for a task can be changed during execution by DETACHing the robot, changing DRY.RUN, and then ATTACHing the robot.

> **NOTE:** Do not allow multiple tasks to change DRY.RUN simultaneously, since the DRY.RUN state could then be different from that expected. Your programs should use a software interlock in this case.

The DRY.RUN switch is initially **disabled**.

WARNING: Digital and analog I/O is not affected by DRY.RUN, so external devices driven by analog or digital output instructions will still operate.

Related Keywords

DISABLE	(monitor command, see the *V+ Operating System Reference Guide)*
ENABLE	(monitor command, see the *V+ Operating System Reference Guide)*
SWITCH	(monitor command, program instruction, and real-valued function)

Syntax

DURATION time *ALWAYS*

Function

Set the minimum execution time for subsequent robot motions.

Usage Considerations

Unless the ALWAYS parameter is specified, only the next robot motion will be affected.

DURATION 0 ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The DURATION instruction affects the DRIVE instruction but not the DELAY instruction.

The setting of the monitor SPEED command affects the results of the DURATION setting.

The DURATION instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the DURATION instruction causes an error.

Parameters

time Real-valued expression that specifies the minimum length of time (in seconds) that subsequent robot motions will take to be performed (see below).

If the value is zero, robot motions are performed without consideration of their time duration and use only the applicable values for SPEED and ACCEL.

ALWAYS Optional keyword that determines how long the new duration will have an effect.

If ALWAYS is included, the specified duration time applies to all subsequent robot motions (until the duration setting is changed by another DURATION instruction). The specified duration applies only to the next robot motion if ALWAYS is not included.

Details

This instruction sets the minimum execution time for subsequent robot motions. For any motion, the time specified by the DURATION instruction has no effect if the duration setting is less than the time computed by the V+ robot-motion trajectory generator (considering the current motion speed and acceleration settings). However, if the duration is longer than the time computed by the trajectory generator, the motion is slowed so that its elapsed time corresponds approximately to the specified duration.

> **NOTE:** Actual motion times may differ slightly from the duration setting due to quantization effects and due to acceleration and deceleration profiling.

The duration instruction does not specify the duration of an entire motion but instead specifies the minimum time of the constant-velocity segment plus one-half the acceleration and deceleration segments. In this way, continuous-path motions (in which individual motions are blended together) get the correct duration, but a single motion takes *longer* than the specified duration. In other words, the time of motion is primarily defined either by the value of DURATION or SPEED, using whichever value gives the longer time.

This instruction is very useful. Consider, for example, a situation where the value of a periodic, external signal is employed to continuously correct the path of the robot while the robot is moving. The DURATION instruction can be used to match the motion execution time to the sensor sampling rate and processing time. This ensures that the robot will be kept in motion while new information is being processed. A sample program of this type is shown later.

Example

The following example reads an external sensor and moves to the computed robot location. This sequence is repeated 20 times at intervals of 96 milliseconds[1] (6/TPS seconds). Note that the motion speed is set to a very large value to make sure the motion is paced by the duration setting.

```
DURATION 6/TPS ALWAYS     ;Each motion to be 6 ticks long

SPEED 200 ALWAYS          ;Motion time determined primarily
                          ;by DURATION, not SPEED

FOR i = 1 TO 20           ;Repeat 20 times...
   CALL read.signal(loc) ;Get new step from sensor
   MOVE loc               ;Move to the location
END
```

Related Keywords

ACCEL (program instruction)

DELAY (program instruction)

DURATION (real-valued function)

PAYLOAD (program instruction)

SELECT (program instruction and real-valued function)

SPEED (monitor command and program instruction)

[1] This assumes the default period (tick) of 16 milliseconds for the V+ trajectory generator.

Syntax

```
DURATION (select)
```

Function

Return the current setting of one of the motion DURATION specifications.

Usage Considerations

The DURATION function returns information for the robot selected by the task executing the function.

If the V+ system is not configured to control a robot, the DURATION function will not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

Parameter

`select` Real-valued expression whose value determines which duration value should be returned (see below).

Details

This function returns the user-specified minimum robot motion duration (in seconds) corresponding to the **`select`** parameter value. (See the description of the DURATION program instruction for an explanation of the specification of motion duration times.)

Different **`select`** values determine when the duration time returned will apply, as listed below. (All other values for the **`select`** parameter are considered invalid.)

select	DURATION value returned
2	Permanent minimum robot motion duration (set by a DURATION ... ALWAYS program instruction)
3	Temporary motion duration for the current or last motion
4	Temporary motion duration to be used for the next motion

Related Keywords

DURATION (program instruction)

SELECT (program instruction and real-valued function)

Syntax

```
DX (location)

DY (location)

DZ (location)
```

Function

Return a displacement component of a given transformation value.

Parameter

location	Transformation value from which a component is desired. This can be a function, a variable, or a compound transformation.

Details

These three functions return the respective displacement components of the specified transformation value.

> **NOTE:** The DECOMPOSE instruction can also be used to obtain the displacement components of a transformation value. If the rotation components are desired, that instruction must be used. DECOMPOSE is more efficient if more than one element is needed and the location is a compound transformation.

Example

Consider a transformation start with the following components:

```
125, 250, –50, 135, 50, 75
```

The following function references will then yield the indicated values:

```
DX(start) ;Returns 125.00

DY(start) ;Returns 250.00

DZ(start) ;Returns –50.00
```

Related Keywords

DECOMPOSE (program instruction)

RX RY RZ (transformation functions)

Syntax

ELSE

Function

Separate the alternate group of statements in an IF ... THEN control structure.

Usage Considerations

ELSE can be used only within an IF ... THEN ... ELSE ... END control structure.

Details

Marks the end of a group of statements to be executed if the value of the logical expression in an IF logical_expr THEN control structure is nonzero, and the start of the group of statements to be executed if the value is zero.

See IF ... THEN for more details and examples.

Related Keyword

IF... THEN (program instruction)

Syntax

```
ENABLE switch, ..., switch
```

Function

Turn on one or more system control switches.

Usage Considerations

The ENABLE monitor command can be used when a program is executing.

If a specified switch accepts an index qualifier and the index is zero or omitted (with or without the brackets), *all* the elements of the switch array are enabled.

Parameter

`switch` Name of a system switch to be turned on.

The name can be abbreviated to the minimum length that uniquely identifies the switch. That is, for example, the MESSAGES switch can be referred to with ME, since there is no other switch with a name beginning with the letters ME.

Details

System switches control various aspects of the operation of the V+ system, including some optional subsystems such as vision. The Switch entry in the index for this document directs you to the detailed descriptions of these switches.

Other system switches are available when options are installed. Refer to the option documentation for details. For example, the switches associated with the AdeptVision options are described in the ***AdeptVision Reference Guide***.

When a switch is enabled, or turned on, the feature it controls is functional and available for use. Turning a switch off with the DISABLE monitor command or program instruction makes the associated feature not functional or available for use.

NOTE: The system switches are shared by all the program tasks. Thus, care should be exercised when multiple tasks are disabling and enabling switches. Otherwise, the switches may not be set correctly for one or more of the tasks.

Disabling the DRY.RUN switch does not have effect until the next EXECUTE command or instruction is processed for task #0, an ATTACH instruction is executed for the robot, or a CALIBRATE command or instruction is processed.

The SWITCH monitor command or the SWITCH real-valued function can be used to determine the status of a switch at any time. The SWITCH program instruction can be used, like the ENABLE instruction, to set a switch.

Example

```
ENABLE MESSAGES ;Turns on the MESSAGES switch.
```

Related Keywords

DISABLE (monitor command and program instruction)

SWITCH (monitor command, program instruction, and real-valued function)

Syntax

```
$ENCODE (output_specification, output_specification, ...)
```

Function

Return a string created from output specifications. The string produced is similar to the output of a TYPE instruction.

Parameter

An output specification can consist of any of the following components (in any order) separated by commas:

1. A string expression.
2. A real-valued expression, which is evaluated to determine a value to be displayed.
3. Format-control information, which determines the format of the output message.

Details

This function makes strings normally produced by the TYPE instruction available within a program. That is, $ENCODE does not generate any output, but rather creates a string value.

The following format specifiers can be used to control the display of numeric values. These settings remain in effect for the remainder of the function parameter list unless another specifier is used.

For all these specifiers, if a value is too large to be displayed in the given field width, the field is filled with asterisk characters (*).

/D	Use the default format, which is equivalent to /G15.8 (see below), except trailing zeros and all but one leading space are omitted.

The following format specifications accept a zero as the width field. That causes the actual field size to vary to fit the value, and causes all leading spaces to be suppressed. That is useful when a value is displayed within a line of text, or at the end of a line.

/En.m	Format values in scientific notation (for example, –1.234E+2) in fields n spaces wide with m digits in the fractional parts. (If n is not 0, its value must be at least five larger than the value of m.)

/Fn.m	Format values in fixed-point notation (for example, –123.4) in fields n spaces wide, with m digits in the fractional parts.
/Gn.m	Format values in F format with m digits in the fractional parts if that can be done in fields n spaces wide. Otherwise use /En.m format.
/Hn	Format values as hexadecimal integers in fields n spaces wide.
/In	Format values as decimal integers in fields n spaces wide.
/On	Format values as octal integers in fields n spaces wide.

The following specifiers can be used to insert special characters in the string:

/Cn	Include the characters carriage return (CR) and line feed (LF) n times.

If the string resulting from the $ENCODE function is output to the terminal, this will result in n blank lines if the control specifier is at the beginning or end of the function parameter list; otherwise n –1 blank lines will result.

/Un	Include the characters necessary to move the cursor up n lines if the resulting string is output to the terminal. (This will work correctly only if the TERMINAL parameter is correctly set for the terminal being used.)
/Xn	Include n spaces.
/B	Include a character that will beep the terminal if the resulting string is output to the terminal.

Example

The program statement:

```
$output = $output+$ENCODE(/F6.2, count)
```

adds a formatted representation of the value of count to the string contained in $output.

The $ENCODE function provides a way of adding format control to the output from PROMPT instructions. This is shown by the following example, in which the value of motor is displayed as part of the prompt message to the operator.

```
PROMPT $ENCODE(/B,"Start motor #",/I0,motor," (Y/N)? "),
$answer
```

This PROMPT instruction will beep the terminal (/B), and display the following user prompt when the value of motor is 3:

```
Start motor #3 (Y/N)?
```

Related Keyword

TYPE (program instruction)

Syntax

```
END
```

Function

Mark the end of a control structure.

Usage Considerations

Every END instruction must be part of a CASE, FOR, IF, or WHILE control structure.

Details

Every CASE, FOR, IF, and WHILE control structure must have its end marked by an END instruction. The V+ editor will display an error message when program editing is exited if there is not the correct number of END instructions in a program (that is, if there are too few or too many).

Related Keywords

CASE	(program instruction)
FOR	(program instruction)
IF ... THEN	(program instruction)
WHILE	(program instruction)

Syntax

```
.END
```

Function

Mark the end of a V+ program.

Usage Considerations

The V+ editors automatically add this line to the end of every program.

Details

Normally, you will not need to concern yourself with the .END step of programs—it is created automatically by the V+ editors. The only time you will see this step while working with the V+ system is when you issue a LISTP monitor command. Then you will see an .END step as the last step of each program.

The .END is important, however, when a program is created on another computer for transfer to a V+ system. In that case, the programmer must be sure to include a line starting with .END at the end of each program (the remainder of the line is ignored by V+). Programs missing the .END instruction will not load correctly into the V+ system.

Related Keyword

.PROGRAM (program instruction)

Syntax

ERROR (task_num, select**)**

Function

Return the error number of a recent error that caused program execution to stop or caused a REACTE reaction.

Parameters

task_num	Real value, variable, or expression (interpreted as an integer) whose value selects the source of the error code as follows:
–1	Return the number of the most recent error from the program in which the ERROR function is executed.
0	Return the number of the most recent error from the program executing as task #0.
>0	Return the number of the most recent error from the program executing as the corresponding task number.
select	Optional real value, variable, or expression (interpreted as an integer) that selects the error information to be returned. (The value 0 is assumed if this parameter is omitted.)
0	Return the error number of the most recent program execution error (excluding I/O errors—see IOSTAT) for the specified program task.
1	If the most recent error (for the specified program task) had an error code less than or equal to –1000 (that is, –1000, –1001, etc.), return the variable portion of the corresponding error message. The value returned should be interpreted as a 6-bit numeric value—see below for details. Zero is returned if the error did not have a variable portion in its message. (Also see select = 3 below.)
2	Return the error number of the most recent error from an MCS instruction executed by the specified program task.
3	Return the number of the robot associated with the most recent error for the specified program task. Zero is returned if the error was not associated with a specific robot. (Also see select = 1.)
4	Return the error code corresponding to any pending RSC errors for the robot selected by the specified user task.

RSC errors may not be valid until 0.1 seconds after a REACTE routine receives an *E-STOP from robot* (–640) error. The REACTE routine can take any time-critical action, wait 0.1 seconds, and then use ERROR(task,4) to get a more definitive error number. If no RSC errors are pending, this function returns zero.

Details

This function is especially useful in a REACTE subroutine program to determine why the REACTE was triggered.

> **NOTE:** The ERROR function does not report errors reported by the IOSTAT function.

Appendix B contains a list of all the V+ error messages and their error numbers.

As noted above, when the select parameter is 1, the value returned by this function should be interpreted as a 6-bit numeric value. The following program illustrates how the value should be interpreted.

```
.PROGRAM error.string(code, vcode, robot, $msg)

; ABSTRACT: Return error message corresponding to
; error code(s).
;
; INPUT PARAMS:
;     code   Basic error code e.g., from
;              ERROR(n) or IOSTAT(lun)
;     vcode  Variable part of the error
;              code [e.g., from ERROR(n,1)].
;     robot  Number of the robot associated
;              with the error [i.e., from
;              ERROR(n,3)].
;
; OUTPUT PARAM:
;     $msg   Corresponding error message may be null

AUTO i, n

$msg = ""               ;Assume no error

IF code < 0 THEN        ;If there was an error...

   $msg = $ERROR(code)  ;Get base message string
```

```
IF (-1100 < code) AND (code <= -1000) THEN
;Bit field
   n = 1                          ;Initialize bit mask
   FOR i = 1 TO 7                 ;For each of 7 bits
     IF vcode BAND n THEN         ;If this bit is set,
       $msg = $msg+$ENCODE(i)     ;add it to message
     END
     n = 2*n                      ;Shift the mask 1 bit
   END
END
IF (-1200 < code) AND (code <= -1100) THEN
   $msg = $msg+$ENCODE(vcode)     ;Add number
END

IF robot AND (SELECT(ROBOT,1) > 1) THEN
;Maybe add
   $msg = $msg+" (Robot"+$ENCODE(robot)+")"
;robot number
END

.END
```

Related Keywords

$ERROR (string-valued function)

IOSTAT (real-valued function)

REACTE (program instruction)

Syntax

```
$ERROR (error_code)
```

Function

Return the error message associated with the given error code.

Parameter

error_code Real-valued expression with a negative value, that identifies an error condition.

Details

All the error codes returned by the IOSTAT function and by the ERROR real-valued function can be converted into their corresponding V+ error message strings with this function. (The ERROR real-valued function must be used to determine the *variable* portion of the error message for an error code less than or equal to –1000.)

Appendix B contains a list of all the V+ error messages and their error codes.

Example

The following program segment displays an error message if an I/O error occurs:

```
READ (5) $input
IF IOSTAT(5) < 0 THEN
   TYPE "I/O error during read: ", $ERROR(IOSTAT(5))
   HALT
END
```

Related Keywords

ERROR (real-valued function)

IOSTAT (real-valued function)

REACTE (program instruction)

Syntax

ESTOP

Function

Assert the emergency-stop signal to stop the robot.

Details

This instruction immediately asserts the backplane emergency-stop signal, provided that a later model SIO module is installed. It immediately deasserts High Power Enable (HPE) and then proceeds with a normal power-down sequence.

Related Keywords

BRAKE	(program instruction
PANIC	(monitor command, see the *V+ Operating System Reference Guide)*
STATE	(real-valued function)

Syntax

```
     EXECUTE /C task_num program(param_list), cycles, step,
priority[i]
```

Function

Begin execution of a control program.

Usage Considerations

A program cannot already be active as the specified program task.

The `priority` values are normally used only during experimental tuning of the system execution tasks.

Parameters

`/C` Optional qualifier that conditionally attaches the selected robot. The qualifier has an effect only when starting the execution of task 0.

`task_num` Optional real value or expression specifying which program task is to be activated. (See below for the default. See the *V+ Language User's Guide* for information on tasks.)

`program` Name of the program to be executed. If the name is omitted, the last program name specified in an EXECUTE command or instruction (or PRIME command) for the selected task is used.

WARNING: Entering an EXECUTE instruction with no program specified can result in unexpected motion of the robot because the previous program is executed again.

`param_list` Optional list of constants, variables, or expressions separated by commas, that must correspond in type and number to the arguments in the .PROGRAM statement for the program specified. If no arguments are required by the program, the list is blank, but the parentheses must be entered.

Program parameters may be omitted as desired, using commas to skip omitted parameters. No commas are required if parameters are omitted at the end of the list. Omitted parameters are passed to the called program as undefined and can be detected with the DEFINED real-valued function.

Automatic variables (and subroutine arguments) cannot be passed by reference in an EXECUTE instruction. They must be passed by value (see the description of CALL).

The parameters are evaluated in the context of the new task that is started (see below).

`cycles` Optional real value, variable, or expression (interpreted as an integer) that specifies the number of program execution cycles to be performed. If omitted, the cycle count is assumed to be 1. For unlimited cycles, specify any negative value. The maximum loop count value allowed is 32,767.

`step` Optional real value, variable, or expression (interpreted as an integer) that specifies the step at which program execution is to begin. If omitted, program execution begins at the first executable statement in the program (that is, after the initial blank and comment lines and all the AUTO and LOCAL instructions).

`priority[]` Optional array of real values (interpreted as integers) that are used by V+ to override the default execution priority within the task. The array contains 16 elements, which specify the program priority in each of 16 one-millisecond time slices. If specified, the elements must be in the range –1 to 64, inclusive. (See the ***V+ Language User's Guide*** for the details of task scheduling.)

`i` Optional real value, variable, or expression (interpreted as an integer) that specifies the index value of the first element.

Details

This command initiates execution of the specified control program. The program will be executed `cycles` times, starting at the specified program step. If no program is specified, the system reexecutes the last program executed by the selected program task.

Note that EXECUTE can be used as a program instruction as well as a monitor command. Thus, one control program can initiate execution of other, related control programs. (After a program initiates execution of another program, the initiating program can use the STATUS and ERROR real-valued functions to monitor the status of the other program.)

If the task number is not specified and the system is not in DEBUG mode, the EXECUTE command accesses task number 0. The current debug task is assumed if the system is in DEBUG mode. The EXECUTE instruction always accesses task #0 if the task number is omitted.

The optional /C qualifier has an effect only when starting execution of task 0. When /C is not specified, an EXECUTE instruction for task 0 fails if the robot cannot be attached; attachment requires that the robot be calibrated and that arm power be enabled (or that the DRY.RUN switch is enabled). When /C is specified, an EXECUTE instruction for task 0 attempts to attach the robot but allows execution to continue without any indication of error if the robot cannot be attached.

Certain default conditions are assumed whenever program execution is initiated. They are equivalent to the following program instructions:

```
DURATION 0 ALWAYS

FINE 100 ALWAYS

LOCK 0

MULTIPLE ALWAYS

NULL ALWAYS

SPEED 100,100 ALWAYS

SELECT ROBOT = 1
```

Also, the robot configuration is saved for subsequent motions.

An execution cycle is terminated when a STOP instruction is executed, a RETURN instruction is executed in the top-level program, or the last defined step of the program is encountered. The value of `cycles` can range from –32,768 to 32,767. The program is executed one time if `cycles` is omitted or has the value 0 or 1. Any negative value for `cycles` causes the program to be executed continuously until a HALT instruction is executed, an error occurs, or the user (or another program) aborts execution of the program.

> **NOTE:** Each time an execution cycle is initiated, the execution parameters are reset to their default values. This includes motion speed, robot configuration, and servo modes. However, the robot currently selected is not changed.

If `step` is specified, the program begins execution at that step for the first pass. Successive cycles *always* begin at the first executable step of the program.

The `priority` values override the default execution configuration for the specified program task. The specified priorities remain in effect only until the next time an EXECUTE instruction is issued for the program task. The acceptable values are:

–1	do not run in this slice even if no other task is ready to run
0	run in this task only when no other task is ready to run
1-64	run in this task according to the specified priority (64 is the highest priority; higher priority tasks may lock out lower priority tasks for the duration of the time slice)

When setting priorities, remember:

1-31	are normal user task priorities
32-62	are used by V+ device drivers and system tasks
63	is used by the trajectory generator. Do not use 63 or 64 unless you have very short task execution times or jerky robot motions may result.

The `priority` values can have a very significant influence on the performance of the system. Thus they should be specified very carefully. Normally, these parameters will be used only when experimenting with execution configurations in preparation for changing the default configuration recorded on the V+ system file. (See the ***V+ Language User's Guide*** for a discussion of task execution configuration and for information on how to change the default configuration permanently.)

All the instruction parameters are evaluated in the context of the *new* task that is started. This can lead to unexpected results when the EXECUTE program instruction is used, and an attempt is made to pass a task-dependent value (for example, the TASK real-valued function). In such a case, if you want the task-dependent value to reflect the *invoking* task, you must assign the task-dependent value to a variable and pass that variable.

Examples

Initiate execution (as task #0) of the program named assembly, with execution to continue indefinitely (that is, until execution is aborted, a HALT instruction is executed, or a run-time error occurs):

```
EXECUTE assembly, -1
```

Initiate execution, with program task #2, of the program named test. The parameter values 1 and 2 are passed to the program.

```
EXECUTE 2 test(1,2)
```

Initiate execution of the last program executed by program task #0 (or by the current debug task). No parameters are passed to the program.

```
EXECUTE
```

The following program segment shows how an application program can be initiated from another application program (the ABORT and CYCLE.END program instructions are used to make sure the specified program task is not already active):

```
ABORT 3                ;Abort any program already active
CYCLE.END 3            ;Wait for execution to abort
EXECUTE 3 new.program ;Start up the new program
```

Related Keywords

ABORT	(monitor command and program instruction)
CALL	(program instruction)
CYCLE.END	(monitor command and program instruction)
KILL	(monitor command and program instruction)
PRIME	(monitor command)
PROCEED	(monitor command)
RETRY	(monitor command)
SSTEP	(monitor command)
STATUS	(monitor command and real-valued function)
XSTEP	(monitor command)

See the *V+ Operating System Reference Guide* for details on monitor commands.

Syntax

EXIT count

Function

Branch to the statement following the nth nested loop of a control structure.

Usage Considerations

This instruction works with the FOR, WHILE, and DO control structures.

Parameter

count	Integer value (expressions and variables are not acceptable) specifying how many nested structures to exit. Default is 1.

Details

When an EXIT instruction is reached, the control structure is terminated and execution continues at the first instruction following the outermost control structure exited.

Example

If input signal 1001 is set, exit one control structure; if 1002 is set, exit three control structures:

```
27 FOR i = 1 to 40
28   WHILE ctrl.var DO
29     DO
30       IF SIG(1002) THEN
31         EXIT 3 ;Exit to step 39
32       END
33       IF SIG(1001) THEN
34         EXIT  ;Exit to step 37
35       END
36     UNTIL FALSE
37   END
38 END
```

Related Keywords

DO... UNTIL	(program instruction)
FOR	(program instruction)
NEXT	(program instruction)
WHILE...DO	(program instruction)

Syntax

FALSE

Function

Return the value used by V+ to represent a logical false result.

Usage Considerations

The word false cannot be used as a program name or variable name.

Details

This named constant is useful for situations where true and false conditions need to be specified. The value returned is 0.

Example

The following program loop will execute continuously until the subroutine cycle returns a FALSE value for the real variable **continue**:

```
DO
   CALL cycle(continue)
UNTIL continue == FALSE
```

Related Keywords

OFF (real-valued function)

TRUE (real-valued function)

Syntax

```
FCLOSE (logical_unit)
```

Function

Close the disk file, graphics window, or graphics icon currently open on the specified logical unit.

Usage Considerations

No error is generated if a file or graphics window is not open on the logical unit, although the IOSTAT real-valued function will return an error code.

When a graphics window is closed, the window is not deleted from graphics memory and its stacking and display status are not changed.

Parameter

`logical_unit` Real value, variable, or expression (interpreted as an integer) that identifies the device to be accessed. (See the ATTACH instruction for a description of logical unit numbers.)

Details

After a program has finished accessing a file that has been opened via an FOPEN_ instruction, the program must close the file by executing an FCLOSE instruction. FCLOSE frees the file for access by the V+ monitor and other programs. In addition, for files that have been opened for writing, FCLOSE writes out any data still buffered by V+ and updates the file directory information. Thus, if this operation is not performed, the disk file may not actually contain all of the information written to it.

If a program is finished accessing a graphics window, or needs to reuse its logical unit number, the window can be closed with this instruction. After a window is closed, it can be deleted with an FDELETE instruction or it can be opened again later with an FOPEN instruction. (Note that reopening a window resets all its text and graphics attributes [for example, color, font ID, character path and orientation, texture, logical operation, and enabled events], which must be explicitly reset by the program before attempting output to the window.)

An FCLOSE operation is automatically performed on a logical unit when the unit is detached, when the program that issued the FOPEN_ completes execution, or when a KILL of the program task is performed.

The IOSTAT real-valued function should be used to check for successful completion of a close operation. (The error code for File not opened will be returned if there was no file or window currently open on the specified logical unit.)

Related Keywords

ATTACH	(program instruction)
DETACH	(program instruction)
FOPEN	(program instruction)
FOPEN_	(program instructions)
IOSTAT	(real-valued function)
KILL	(monitor command and program instruction)

Syntax

```
FCMND (logical_unit, command_code) $out_string, $in_string
```

Function

Generate a device-specific command to the input/output device specified by the logical unit.

Usage Considerations

The logical unit referenced must have been previously attached.

As appropriate, the current default device, unit, and directory path are considered for any disk file specification (see the DEFAULT command).

Parameters

`logical_unit` Real-valued expression that identifies the device to be accessed. (See the ATTACH instruction for a description of logical unit numbers.)

`command_code` Real-valued expression that specifies the command to be executed. (See the explanation of command codes below.)

`$out_string` String constant, variable, or expression that is transmitted to the device along with the command code to specify the operation to be performed.

`$in_string` Optional string variable. This variable receives any information returned from the device as a result of the command.

Details

This instruction allows a program to generate device-specific command sequences. For example, this instruction can be used to send a command to the disk to delete a file or to rename a file. Since these are maintenance operations, which are not generally performed by V+ programs, no special-purpose V+ program instructions exist for performing these operations.

Command Codes

At present, the FCMND instruction can be used only in connection with the storage disks, the USER serial communication ports, and graphics windows. The command codes listed below are accepted.

6 Rename a file. The `$out_string` parameter must contain the new name of the file (including any required disk unit and directory path specification), followed by a space, followed by the old file name (which cannot include a disk or directory specification). Note that the data contained in `$out_string` is *not* the same as that of the argument list of the FRENAME monitor command. No data is returned in the `$in_string` variable.

7 Compress the disk. This command is invalid for local disks.

8 Format the disk. The `$out_string` parameter must contain the name of the disk unit to format, followed by any required qualifiers. The data contained in `$out_string` must be identical to that of the argument list of a FORMAT monitor command. On completion, the `$in_string` variable will contain text indicating how many bad blocks were located.

CAUTION: Formatting a disk erases all the information on the disk.

14 Create a subdirectory. The `$out_string` parameter must contain the specification of the subdirectory, including an optional unit name if the current default disk unit is not to be accessed. (Refer to the ***V+Operating System User's Guide*** for a description of subdirectory specifications.)

NOTE: Only the final subdirectory in the specified directory path is created by this operation. That is, all the intermediate subdirectories must already exist, and they are not created.

15 Delete a subdirectory. The `$out_string` parameter must contain the specification of the subdirectory, including an optional unit name if the current default disk unit is not to be accessed. (See the ***V+Operating System User's Guide*** for a description of subdirectory specifications.)

NOTE: Only the final subdirectory in the specified directory path is deleted by this operation. That is, all the intermediate subdirectories must already exist, and they are not deleted.

19 Assert the creation date/time for the file currently open on the specified logical unit. This command can be issued at any time a disk file is opened. Once asserted, when the file is closed, the file's creation date and time are set equal to the specified values rather than the current date and time. Also, if this command is issued when the file is closed, V+ does not automatically assert the not archived bit. The input string must contain **date** and **time,** where

date is a 16-bit integer word representing the date in the standard compressed format used by the TIME and $TIME functions.

time is a 16-bit integer word representing the time in the standard compressed format.

20 Return the number of unused and total number of sectors on a local disk. The returned string is in the form `uuuu/ttttt` where `uuuuu` is the number of unused sectors and `ttttt` is the total number of sectors. A file must be open on the drive (with prereads disabled). The open file identifies the disk unit.

21 Read the creation date/time for the file currently open on the specified logical unit. This command can be issued any time after a file has been opened. Normally, this command returns the values that are read from the disk directory at the time the file was opened. However, if an FCMND 19 instruction has been issued to assert file creation date and time, FCMND 21 returns the value set by FCMND 19. The string returned by this command contains **date** and **time** (use INTB to extract the values), where

date is a 16-bit integer word representing the date in the standard compressed format used by the TIME and $TIME functions.

time is a 16-bit integer word representing the time in the standard compressed format.

102 Clear the type-ahead buffer for a serial line, or clear the event queue for a graphics window. This command, which is recognized only by the serial communication lines and the graphics logical units, does not process any arguments.

106 Read modem control flags. The flags are returned as a one-byte string with the following interpretation:

Bit	State of:
1	Request to Send (RTS)
2	Data Terminal Ready (DTR)
5	Input Clear to Send (CTS)
6	Input Data Set Ready (DSR)

500 Return information about DDCMP status. (This FCMND is present in all V+ systems that support DDCMP.) The FCMND reply string may be parsed using INTB and LNGB functions to extract the binary data, as described in the following code. When the instruction

```
FCMND (lun,500) "", $reply
```

is executed, the string variable `$reply` receives packed binary data regarding the DDCMP line attached on the specified logical unit. Then the functions shown in **Table 2-6 on page 215** can be used to extract the data.

Table 2-6. INTB and LNGB Functions

Function	Notes
INTB($reply,1) DDCMP network state (0, 1, or 2)	0 = Line is closed 1 = Line is open but waiting for remote 2 = Line is active
INTB($reply,3)	Not used
LNGB($reply,5)	Local media error count
LNGB($reply,9)	Local timing error count
LNGB($reply,13)	Local format error count
LNGB($reply,17)	Remote media error count
LNGB($reply,21)	Remote timing error count
LNGB($reply,25)	Remote format error count
LNGB($reply,29)	Count of blocks sent
LNGB($reply,33)	Count of blocks received

501 Set DDCMP communication parameters. This command is recognized only by serial communication lines configured for use with the DDCMP protocol. See the ***V+ Language User's Guide*** for the details of this command.

600 Initiate a close connection from the TCP server side for the client identified by the handle number **handle** in the instruction FCMND (lun, 600) $INTB(handle). Note, however, that close-connection requests are more commonly initiated by the client side.

601 Initiate a PING command (see the ***V+ Operating System Reference Guide*** for details on the PING monitor command). The resulting IOSTAT value is either 1, indicating the client was found on the network, or –562, indicating a network timeout.

With the exception of the CLOSE command, a file cannot be open on the logical unit when the FCMND is executed.

Any error in the specification of this instruction (such as attempting to access an invalid unit) will cause a program error and will halt program execution. However, errors associated with performing the actual operations (such as device not ready) do not halt program execution since these errors can occur in the normal operation of a program. These normal errors can be detected by using the IOSTAT function after performing the FCMND. In general, it is good practice always to test whether each FCMND operation completed successfully using IOSTAT.

Examples

Return modem control bit flags for the serial port attached to logical unit 10:

```
FCMND (10,106), $temp
flags = ASC($temp)
```

Format the disk loaded in drive A in double-sided, double-density format and return the string containing the bad-block count in $bad:

```
FCMND (5, 8) "A:/Q", $bad
```

Specify a time-out interval of 2 seconds, with maximums of 20 time-outs and 8 NAK retries.

```
FCMND (lun, 501) $CHR(2)+$CHR(20)+$CHR(8)
```

Check to see if a client is on the network.

```
FCMND (lun, 601) "node_address", $str
```

Related Keywords

ATTACH	(program instruction)
DETACH	(program instruction)
FDELETE	(monitor command and program instruction)
FDIRECTORY	(monitor command
FOPEN_	(program instruction)
FORMAT	(monitor command)
FRENAME	(monitor command)
IOSTAT	(real-valued function)
MCS	(program instruction)

See the *V+ Operating System Reference Guide* for details on monitor commands.

Syntax

```
FDELETE (logical_unit) object
```

Function

Delete the specified disk file, the specified graphics window and all its child windows, or the specified graphics icon.

Usage Considerations

The logical unit number must be attached, but no file or window can be currently open on that logical unit.

The window cannot be deleted if it (or any of its child windows) is open as any other logical unit or by any other program task.

Parameters

`logical_unit` Real value, variable, or expression (interpreted as an integer) that corresponds to a disk or window logical unit. (See the ATTACH instruction for a description of logical unit numbers.)

`object` String constant, variable, or expression specifying the disk file, graphics window, or graphics icon to delete. The error *Nonexistent file* will be reported (via IOSTAT) if the specified object does not exist.

For disk files, the string may contain an optional disk unit and an optional directory path, and must contain a file name, a period (.), and a file extension. The current default disk unit and directory path are considered as appropriate (see the DEFAULT command).

For graphics windows, the string must fully specify the position in the window tree of the window to be deleted.

For graphics icons, the string must specify the name of the icon, followed by /ICON.

Details

If a disk logical unit number is specified, the **`object`** parameter is interpreted as the specification of a disk file to be deleted. If the deletion fails for any reason (for example, the file does not exist, or the disk is protected), an error will be returned via the IOSTAT real-valued function.

NOTE: In order to delete a file from a 3-1/2" diskette, the write-protect slider must be in the position that covers the hole.

If the logical unit number specified is for a graphics window, the **`object`** parameter is interpreted as the specification of a graphics window or icon to be deleted. When a window is specified, that window *and* all of its child windows are deleted. If any of the window's children cannot be deleted, the specified window is not affected and an error is returned (via the IOSTAT real-valued function). When a window is deleted, it is erased from the display. (A window must be FCLOSEd before it can be FDELETEd.)

When a graphics logical unit is accessed, a `*Protection error*` message is reported (via IOSTAT) if a system window or icon is specified.

NOTE: The DISKCOPY utility may provide a more convenient way of deleting multiple files.

Examples

Delete the disk file defined by the file specification in the string variable $file:

```
FDELETE (5) $file
```

Delete the top-level window named TEST, and all of its child windows. The logical unit defined by main must be a graphics logical unit:

```
FDELETE (main) "TEST"
```

Delete the graphics window named ERROR, which is a child of the top-level window named VISION:

```
FDELETE (21) "VISION\ERROR"
```

Delete the graphics icon named BUTTON:

```
FDELETE (20) "BUTTON/ICON"
```

Related Keywords

ATTACH	(program instruction)
FCLOSE	(program instruction)
FDELETE	(monitor command, see the *V+ Operating System Reference Guide)*
FOPEN	(program instruction)
IOSTAT	(real-valued function)

Syntax

```
FEMPTY (logical_unit)
```

Function

Empty any internal buffers in use for a disk file or a graphics window by writing the buffers to the file or window if necessary.

Usage Considerations

When accessing a file, the file must be open for *random access* on the specified logical unit (see the FOPEN_ instructions).

When accessing a graphics window, this instruction is useful only for a window that is opened in buffered mode. (That is, the /BUFFERED attribute was specified in the FOPEN instruction that opened the window.)

Parameter

`logical_unit` Real value, variable, or expression (interpreted as an integer) that identifies the device to be accessed. (See the ATTACH instruction for a description of logical unit numbers.)

Details

During random-access I/O of a disk file, V+ writes data to the disk in blocks of 512 bytes (characters). For efficiency, when a record with a length of less than 512 bytes is written using a WRITE instruction, that data is stored in an internal buffer and might not actually be written to the disk until a later time.

When a disk logical unit is referenced, the FEMPTY instruction directs V+ to write its internal buffer contents immediately to the disk file. That is useful, for example, in applications where data integrity is especially critical (see FOPEN_ for details on defeating buffering).

When a window logical unit is referenced, the FEMPTY instruction forces all buffered graphics output to be immediately written to the window.

The IOSTAT real-valued function can be used to determine if any error results from an FEMPTY operation.

Examples

Empty the internal output buffer for logical unit 5 and write it to the disk immediately:

```
FEMPTY (5)
```

Empty the internal buffer for graphics logical unit 20 by writing it to the window immediately:

```
FEMPTY (20)
```

Related Keywords

ATTACH	(program instruction)
FOPEN	(program instruction)
FOPEN_	(program instruction)
GARC	(program instruction)
GCHAIN	(program instruction)
GCLEAR	(program instruction)
GCLIP	(program instruction)
GCOPY	(program instruction)
GFLOOD	(program instruction)
GICON	(program instruction)
GLINE	(program instruction)
GLINES	(program instruction)
GPANEL	(program instruction)
GPOINT	(program instruction)
GRECTANGLE	(program instruction)
GSCAN	(program instruction)
GSLIDE	(program instruction)
GTYPE	(program instruction)
IOSTAT	(real-valued function)
WRITE	(program instruction)

Syntax

```
FINE tolerance ALWAYS
```

Function

Enable a high-precision feature of the robot hardware servo.

Usage Considerations

Only the next robot motion will be affected if the ALWAYS parameter is not specified.

If the `tolerance` parameter is specified, its value becomes the default for any subsequent FINE instruction executed during the current execution cycle (regardless of whether ALWAYS is specified).

This is the default state of the V+ system. FINE 100 ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The FINE instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the FINE instruction causes an error.

Parameters

`tolerance`	Optional real value, variable, or expression that specifies the percentage of the standard fine tolerances that are to be used for each joint of the robot attached by the current execution task.
`ALWAYS`	Optional qualifier that establishes FINE as the default condition. That is, FINE will remain in effect continuously until disabled by a COARSE instruction. If ALWAYS is not specified, the FINE instruction will apply only to the next robot motion.

Details

Enables the high-precision feature in the robot motion servo system so that only small errors in the final positions of the robot joints are permitted at the ends of motions. This produces high-accuracy motions but increases cycle times since the settling time at the end of each motion is increased.

If the `tolerance` parameter is specified, the new setting takes effect at the start of the next motion. Also, the value becomes the default for any subsequent FINE instruction executed during the current execution cycle (regardless of whether or not ALWAYS is specified). (Changing the FINE tolerance does not affect the COARSE tolerance.)

If the `tolerance` parameter is omitted, the most recent setting (for the attached robot) is used. The default setting is restored to 100 percent when program execution begins, or a new execution cycle starts (assuming that the robot is attached to the program).

The actual tolerance achieved when FINE is in effect for an AdeptMotion VME device is set with the SPEC utility program.

Examples

Enable the high-tolerance feature only for the next motion:

```
FINE
```

Enable the high-tolerance feature for the next motion, with the tolerance settings changed to 50% of the standard tolerance for each joint (that is, a tighter tolerance):

```
FINE 50
```

Enable the high-tolerance feature until it is explicitly disabled:

```
FINE ALWAYS
```

Related Keywords

COARSE (program instruction)

CONFIG (real-valued function)

NONULL (program instruction)

NULL (program instruction)

SELECT (program instruction and real-valued function)

Syntax

FLIP

Function

Request a change in the robot configuration during the next motion so that the pitch angle of the robot wrist has a negative value.

Usage Considerations

Configuration changes cannot be made during straight-line motions.

If the selected robot does not support a flip configuration, this instruction is ignored by the robot.

The FLIP instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the FLIP instruction causes an error.

Details

Asserting a FLIP configuration forces the wrist joint to have a positive rotation (top robot in **Figure 2-3 on page 224**). Asserting a NOFLIP configuration forces a wrist joint to have a negative rotation (bottom robot in **Figure 2-3**). Wrist joints angles are expressed as ±180°. Remember, robots can change configuration only during joint-interpolated moves.

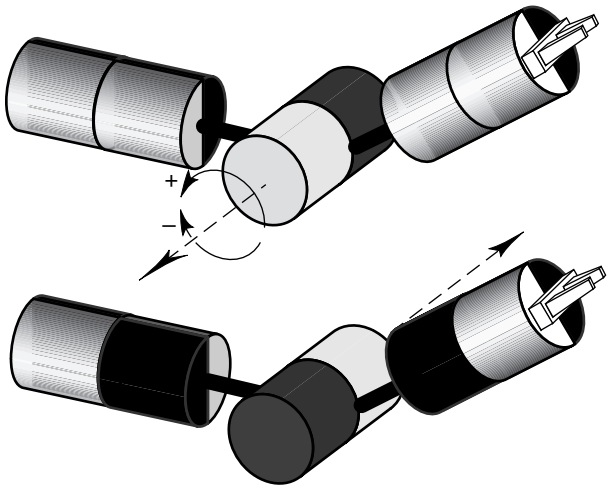


Figure 2-3. FLIP/NOFLIP

Related Keywords

CONFIG	(real-valued function)
NOFLIP	(program instruction)
SELECT	(program instruction and real-valued function)

Syntax

FLTB ($string, first_char**)**

Function

Return the value of four bytes of a string interpreted as an IEEE single-precision floating-point number.

Parameters

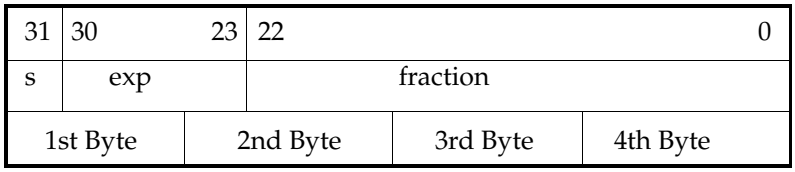
$string String expression that contains the four bytes to be converted.

first_char Optional real-valued expression that specifies the position of the first of the four bytes in the string.

If first_char is omitted or has a value of 0 or 1, the first four bytes of the string are extracted. If first_char is greater than 1, it is interpreted as the character position for the first byte. For example, a value of 2 means that the second, third, fourth, and fifth bytes are extracted. An error is generated if first_char specifies four bytes that are beyond the end of the input string.

Details

Four sequential bytes of the given string are interpreted as being a single-precision (32-bit) floating-point number in the IEEE standard format. This 32-bit field is interpreted as follows:



where:

s is the sign bit, s = 0 for positive, s = 1 for negative.

exp is the binary exponent, biased by –127.

fraction is a binary fraction with an implied 1 to the left of the binary point.

For 0 < exp < 255, the value of a floating-point number is:

$-1^{s} * (1.\text{fraction}) * 2^{exp\ -127}$

For exp = 0, the value is zero; for exp = 255, an overflow error exists.

The main use of this function is to convert a binary floating-point number from an input data record to a value that can be used internally by V+.

Examples

```
FLTB($CHR(^H3F)+$CHR(^H80)+$CHR(0)+$CHR(0)) ;Returns 1.0
FLTB($CHR(^HC0)+$CHR(^H40)+$CHR(0)+$CHR(0)) ;Returns -3.0
```

Related Keywords

ASC (real-valued function)

DBLB (real-valued function)

$DBLB (string-function)

$FLTB (string function)

INTB (real-valued function)

TRANSB (real-valued function)

VAL (real-valued function)

Syntax

$FLTB (value)

Function

Return a 4-byte string containing the binary representation of a real value in single-precision IEEE floating-point format.

Parameter

value	Real-valued expression, the value of which is converted to its IEEE floating-point binary representation.

Details

A real value is converted to its binary representation using the IEEE single-precision standard floating-point format. This 32-bit value is packed as four successive 8-bit characters in a string. See the FLTB real-valued function for a more detailed description of IEEE floating-point format.

The main use of this function is to convert a real value to its binary representation in an output record of a data file.

Example

```
$FLTB(1.215)
;Returns a character string equivalent to:
$CHR(^H3F)+$CHR(^H9B)+$CHR(^H85)+$CHR(^H1F)
```

Related Keywords

$CHR	(string function)
DBLB	(real-valued function)
$DBLB	(string-function)
FLTB	(real-valued function)
$INTB	(string function)
$TRANSB	(string function)

Syntax

FOPEN (logical_unit, mode**) attribute_list**

Function

Create and open a new graphics window or TCP connection, or open an existing graphics window for subsequent input or output.

Open a graphics icon for definition.

Usage Considerations

The logical unit must be attached before an open operation will succeed.

When this instruction is executed, any previous settings (for the specified logical unit) from the following instructions are reset to their default conditions: GCLIP, GCOLOR, GLOGICAL, and GTEXTURE.

Closing or undisplaying a window does not release the graphics memory used by that window. The window must be deleted to release that memory. (See the FDELETE instruction.)

The TCP mode of operation of the FOPEN instruction applies only to Adept controllers with the AdeptNet option and the license for the AdeptTCP/IP Protocol Access.

Parameters

logical_unit Real value, variable, or expression (interpreted as an integer) that defines the logical unit number to be assigned to the window or TCP device. (See the ATTACH instruction for a description of unit numbers.)

mode Optional expression that applies only to TCP logical units and selects the type of TCP connection:
0 = Client mode, 16 = Server mode.

attribute_list List of string constants, variables, and expressions; real values, variables, and expressions; and format specifiers used to assign a name to the window and to define some of the characteristics of the window.

When opening a TCP connection in server mode, this string defines the characteristics of the server. When opening a connection in client mode, the string defines the name of the server in addition to characteristics of the connection.

The attribute list (which is processed like an output specification for the TYPE instruction) is used to compose a single string that is passed to the window manager or TCP driver. The string must begin with the name of the window and can optionally contain keyword attributes that define characteristics of the window. The string must not exceed 512 characters.[1]

The attribute list can consist of one or more components separated by commas. Each component can be expressed in any of the following ways:

1. A string constant, variable, or expression.
2. A real-valued constant, variable, or expression, which is evaluated to determine a value to be used in the control string.
3. A format-control specifier, which determines the format of information in the control string.

Details

There are three basic uses of the FOPEN program instruction:

- **Using FOPEN with Windows**
- **Graphics Memory Allocation**
- **Using FOPEN with TCP**

The following sections describe each of these situations.

Using FOPEN with Windows

The primary use of this instruction is to create a new graphics window or to establish a link to a window that already exists. The information in the instruction attribute list identifies the window and defines its characteristics.

After a window is opened, it may be written to by any of the V+ graphics instructions. In fact, instructions can output to a window even if the window is not displayed. In that case the output will be seen the next time the window is displayed.

[1] A V+ string literal or string variable cannot exceed 128 characters. In order to create an attribute list longer than 128 characters you must concatenate multiple strings .

This instruction can also be used to open a graphics icon for definition in graphics memory. After an icon is opened, data can be written to it with the WRITE instruction (see the GICON instruction for details). The /ICON and /ARRAY attributes, described below, apply to icons.

However, this instruction normally will not be used to define icons. Icons usually will be defined by using the Adept Icon Editing Utility to create icon images and store them in data files on disk.[1] At runtime your application program will call the program load.icon to read the data files and define the icons in graphics memory.

Graphics Memory Allocation

The V+ system has these basic limitations on the number of windows and icons that may be created:

- Graphics memory space
- The number of open windows allowed by the system
- The number of open icons allowed by the system

Graphics memory space is used by all graphics items including windows, fonts, and icons. The portion of graphics memory space that is currently available can be determined with the FREE monitor command or real-valued function. The error *Out of graphics memory* indicates that graphics memory space is filled.

The FREE monitor command and real-valued function can be used to tell how many windows are available. The error *Too many windows* indicates that all the allocated windows are in use. The maximum number of windows is 250.

The maximum number of icons is 255. The error *Too many icons* indicates that all the allocated icon buffers are in use.

Window Name — General Information The first component in the FOPEN argument string must be the window name specifier. This component determines the name of a new or existing window to be opened.

Window names follow a convention similar to that used for disk subdirectories. Like subdirectories, windows can be logically related in tree structures. The backslash character (\) is used to separate names in the tree structure.

[1] The icon editing utility is available with AIM software packages or with the Adept Application Disk. Contact Adept Applications Dept. at (800) 232-3378.

The first name in a window-name specification is for the top-level window. That name must not be preceded by a backslash, but each successive name in the window-name specification must be preceded by a backslash. For example, the name **Graph** could specify a top-level window named Graph. Then, the name **Graph\Dialog** would specify a window named Dialog that is a child of the top-level window Graph.

The same name can be used for more than one window in the tree, except that no two children of one window can have the same name. (When checking for uniqueness, the case of letters in the name is ignored.)

Window names follow the same rules as program and variable names. That is, each name must start with a letter and can contain only letters, numbers, periods, and underline characters. Letters used in window names can be entered in either lowercase or uppercase. Window names can have up to 15 characters.

Note that the name for a top-level, closeable window (see the /CLOSEABLE attribute) is used verbatim in the Adept pull-down menu. Unless a title is explicitly specified for a window (see the /TITLE attribute for the FSET instruction), the name specified in the FOPEN instruction is also used verbatim for the title of the window.

Window Name — Reserved Names There are a few window names that are restricted. The standard windows listed below are defined by the V+ system. These windows cannot be deleted by the user, and programs cannot create top-level windows with the same names.

Name	Description
Monitor	V+ monitor window
Monitor_n	Monitor windows for additional V+ systems (n = 2 to the number of V+ systems)
Status	White bar at the top of the screen
Vision	Vision video and graphics window (*)

* The vision window appears only if the vision option is installed.

If a window has associated control bars, they are accessed as subwindows with the predefined names listed below.

Name	Description
Horz_scroll_bar	Horizontal scroll bar
Menu_bar	Pull-down menu bar
Title_bar	Title bar
Vert_scroll_bar	Vertical scroll bar

Attributes — General Information In addition to the window-name specification, the `attribute_list` parameter can contain a number of keyword attributes that define characteristics of the window. These attributes fall into two groups. The first group, which can appear only in an FOPEN instruction, is documented below. The second group of attributes can be specified with an FOPEN instruction or with an FSET instruction—these attributes are documented with the FSET instruction.

Window attributes can be specified in any order in the attribute list. If an attribute appears more than once in the same list, the last instance of that attribute supersedes all previous instances. Each attribute is specified by a string with the following general form:

```
/keyword argument_1 ... argument_n
```

All the attributes consist of a forward slash (/) and a keyword, which is optionally followed by an argument list. Arguments for an attribute are separated by spaces (*not* by commas). Attribute keywords may be specified in lowercase or uppercase letters, and they can be abbreviated as long as the abbreviation cannot be confused with another attribute's.

> **NOTE:** To ensure that programs will be compatible with future releases of V+, Adept recommends that keywords not be abbreviated.

Keywords and numeric arguments are entered directly in the attribute list. String arguments must be delimited with a pair of single-quote characters ('...'). Each delimited string argument must be separated from the next one by at least one space. A single-quote character may be specified within a delimited string by including two consecutive single-quote characters.

The following examples show how the different types of attributes can be expressed.

`"/POSITION 400 200"`	Keyword and two explicit numeric arguments
`"/BACKGROUND", bg.color`	Keyword and one numeric argument
`"/SHOW_SCROLL"`	Keyword with no arguments
`"/TITLE_BAR 'Erase' 'Exit'"`	Keyword and two string arguments

Remember, the argument list for the FOPEN instruction is processed as a single long string. Extra spaces in the string are ignored, but tab characters are not allowed. When creating the argument list for an FOPEN instruction, it may be helpful to think in terms of composing an output line with the TYPE or WRITE instruction.

Attributes — Default Settings When a new window is created, the maximum-size attribute must be specified. (That attribute is ignored if it is specified when an existing window is reopened.) However, most of the window attributes do not need to be specified—default settings are assumed for attributes that are not mentioned in the attribute list. Note that the minimum window size for windows with a title bar is 8 x 8.

Unless otherwise specified, the window is created if it does not already exist, and the window is opened for read-write access. The /NEW and /WRITEONLY attributes specify opening modes other than the defaults.

A new window has the following default characteristics:

- It has a title bar.
- It has a menu bar.
- It has horizontal and vertical scroll bars.
- If it is a top-level window, it can be closed with the Close icon.

When a (new or existing) window is opened, it has the following default graphics state unless the FOPEN instruction includes an overriding attribute specification.

Characteristic	Setting
Font	1
Character path	0
Character rotation	0
Foreground color	Black
Background color	Light blue
Texture	Transparent
Texture pattern	Solid
Logical operation	Source
Bit-plane mask	All bit planes

Attributes — Descriptions The window attributes that can be specified with the FOPEN instruction are described in Table 2-7 on page 235. The following information is provided for each attribute:

- Keyword that identifies the attribute

  Some attributes can be either on or off. For each such attribute there are two corresponding keywords. The on keyword directly relates to the attribute (for example, /CLOSEABLE); the off keyword has a NO prefix (for example, /NOCLOSEABLE).

- Description of the effect of the attribute

  For attributes with an on/off character, the on condition is described.

  Note that some attributes cannot be changed after a window is created. The keywords for those attributes are ignored when an existing window is opened.

- Description of any arguments required for the attribute
- Default setting assumed if the attribute is not specified

  NOTE: The window attributes shown in Table 2-7 can be specified only with an FOPEN instruction. There are additional attributes that can be specified either with an FOPEN or with an FSET instruction. Those attributes are documented with the FSET instruction.

Table 2-7. FOPEN Window Attributes

Attribute:	**/ARRAY**
Explanation:	This keyword is meaningful only when the /ICON or /DEFFONT keyword is also present (see below). This keyword specifies the maximum index for the icon being opened for definition on the specified logical unit or the maximum characters in the font.
Argument:	Integer maximum index for the icon or font to be defined (0 - 255; 0 means only one element is available)
Default:	/ARRAY 0
Attribute:	**/BACKGROUND**
Explanation:	Specifies the window background color for a new window. This keyword has no effect if the window already exists.
Argument:	Color number (0 to 15 for 16-color window; 0 or 1 for 2-color window)
Default:	Default: /BACKGROUND 5 (for 16-color window) and /BACKGROUND 0 (for 2-color window)
Attribute:	**/BUFFERED**
Explanation:	Indicates that graphics commands are to be buffered up to 512 bytes rather than being sent individually. The buffer contents are sent to the window whenever the buffer is filled or when an FEMPTY instruction is executed. Buffered mode is much more efficient (and faster) when many graphics commands are sent. Commands within a single buffer are executed indivisibly, which is useful for animation (erasing old and drawing new).
Argument:	None
Default:	Output is not buffered
Attribute:	**/CLOSEABLE and /NOCLOSEABLE**
Explanation:	Indicates the window can be closed with the close icon if it is a top-level window. Also, if it is a top-level window, the window name (not title) will appear in the Adept pull-down menu at the top of the screen. (After the pull-down becomes full, all subsequent windows created will be forced to be noncloseable.)
Argument:	None
Default:	/CLOSEABLE

Table 2-7. FOPEN Window Attributes (Continued)

Attribute:	**/COLORS**
Explanation:	Specifies the number of colors (and therefore the number of bits per pixel) for a new window. Two-color windows take only one fourth as much memory as 16-color windows. Windows with 2 colors are always black and light blue. This keyword has no effect if the window already exists.
Argument:	Number of colors (2 or 16 [values 3 through 15 are considered to be 16; values less than 2 or greater than 16 are invalid])
Default:	/COLORS 16
Attribute:	**/DEFFONT n**
Explanation:	Indicates that the logical unit is to be opened for defining a font rather than for accessing a window. (After having been opened for defining a font, the logical unit must be closed [with an FCLOSE instruction] before it can be reopened for accessing a window.) If the font already exists and is non-deletable (e.g., all standard fonts), a *Protection error* is returned. If the font already exists and is deleteable, it is deleted and replaced by the new font. The maximum size of characters is 255 pixels (/SIZE 255 255). If the /POSITION keyword is specified, its arguments specify the position of the reference point of the icon relative to the top left corner of the icon. The arguments must be in the range 0 to 255; their default values are both zero. If the /ARRAY keyword is specified, its argument specifies the maximum index to be used for an array of characters. The value must be in the range 0 to 255.
Argument:	Integer indicating which font is to be defined
Default:	Open a window

Table 2-7. FOPEN Window Attributes (Continued)

Attribute:	**/ICON**
Explanation:	Indicates that the logical unit is to be opened for defining an icon rather than for accessing a window. (After having been opened for defining an icon, the logical unit must be closed [with an FCLOSE instruction] before it can be reopened for accessing a window.) When a logical unit is opened for defining an icon, the name specified with the FOPEN instruction is used for the name of the icon. If the icon already exists, the icon size and maximum index specified in the FOPEN instruction must match the old icon size and maximum index (see below). System icons may not be redefined. The /SIZE keyword must be specified if /ICON is specified. The arguments specified with the /SIZE keyword indicate the size of the icon. Their values must be in the range 1 to 255 (and they are not rounded). If the /POSITION keyword is specified, its arguments specify the position of the reference point of the icon relative to the top left corner of the icon. The arguments must be in the range 0 to 255; their default values are both zero. If the /ARRAY keyword is specified, its argument specifies the maximum index to be used for an icon array. The value must be in the range 0 to 255; the default is zero, indicating only a single icon. See the GICON instruction for information on how to define the image of an icon.
Argument:	None
Default:	Open a window
Attribute:	**/MAXSIZE**
Explanation:	Specifies the size of the window when it is created; also specifies the maximum size the window can be dragged to with the mouse. This keyword has no effect if the window already exists. The X value is always rounded up to be an even number. For windows with a title bar, X and Y values less than 64 are set to 64. For windows without a title bar, X and Y values less than 8 are set to 8.
Argument:	1. X dimension in pixels (if the value is odd, it is rounded up to an even number) 2. Y dimension in pixels
Default:	None—both arguments must be given

Table 2-7. FOPEN Window Attributes (Continued)

Attribute:	**/NEW**
Explanation:	Forces creation of a new window. An error is returned (via IOSTAT) if the window already exists.
Argument:	None
Default:	New or existing window is opened
Attribute:	**/SCROLL_BAR and /NOSCROLL_BAR**
Explanation:	Indicates the window is to be created with vertical and horizontal scroll bars. This keyword has no effect if the window already exists. (Also see /SHOW_SCROLL.)
Argument:	None
Default:	/SCROLL_BAR
Attribute:	**/SHOW_SCROLL and /NOSHOW_SCROLL**
Explanation:	Indicates the window is to be initially displayed with its scroll bars showing. This keyword has no effect if the window already exists. (Also see /SCROLL_BAR.)
Argument:	None
Default:	/NOSHOW_SCROLL
Attribute:	**/TITLE_BAR and /NOTITLE_BAR**
Explanation:	Indicates the window is to be created with a title bar. This keyword has no effect if the window already exists.
Argument:	None
Default:	/TITLE_BAR
Attribute:	**/WRITEONLY**
Explanation:	Indicates window is to be open only for writing (thus, the window will not have an event queue). A window can be simultaneously opened for write-only access by more than one program task. Only one window can have read access at any one time.
Argument:	None
Default:	Window has read access

Using FOPEN with TCP

A TCP/IP *connection* can be opened in either *server mode* or *client mode*. In server mode, one or more clients (depending on the value assigned to /CLIENTS) are allowed to connect to the server for subsequent communication.

To establish a client-server connection, the client must know the port number for the server. For this reason, when using the FOPEN instruction for opening a server connection, the port is explicitly defined using the /LOCAL_PORT attribute. Note that the server does *not* need to know the port number used by the client.

Port numbers 0 through 255 are used by standard TCP application packages. For example, FTP uses ports 20 and 21. By convention, if you are writing your own custom protocol, use a port number greater than 255.

Table 2-8 on page 240 shows valid TCP attributes for the FOPEN instruction.

Table 2-8. FOPEN TCP Attributes

Attribute:	**/BUFFER_SIZE**
Explanation:	Defines the size of the buffer. If omitted, the buffer size defaults to 1024 bytes.
Attribute:	**/CLIENTS**
Explanation:	Defines the number of client connections allowable on a server. If omitted, a single client connection is assumed.
Attribute:	**/LOCAL_PORT**
Explanation:	Defines the local port number for the connection. If omitted, a local port number is automatically assigned.
Attribute:	**/REMOTE_PORT**
Explanation:	Defines the port number of a server to which a client connection is to be made. This *must* be provided when establishing a client connection.

Examples

The example below opens a new or existing window named My_Graph on the logical unit identified by the value of lun. If the window is being created, the size of the window is specified by the variables x.size and y.size; the initial position is fixed at X=400, Y=200; the background color is defined by the value of bckgrnd.col; and the scrollbars are to be displayed. Output to the window is to be buffered. The window will be accessible in read-write mode. (In addition, there are several other attributes that will have default settings—see the FSET instruction.)

> **NOTE:** The FOPEN instruction is shown on multiple lines because of the limited page width. In a V+ program the instruction must be fully contained on a single line.

```
FOPEN (lun) "My_Graph", "/MAXSIZE", x.size, y.size,
"/POSITION 400 200 /BACKGROUND", bckgrnd.col,
"/SHOW_SCROLL/BUFFER"
```

Assuming that x.size is 100, y.size is 150, and bckgrnd.col is 4, the string below will result when the argument list is processed. (Note that spaces between attributes are optional, but the spaces between keywords and parameters or those between parameters are not.)

```
"My_Graph/MAXSIZE 100 150/POSITION 400 200 /BACKGROUND 4
/SHOW_SCROLL/BUFFER"
```

In this case, the following attribute settings will be used by default:

```
"/COLORS 16 /CLOSEABLE /SCROLL_BAR /TITLE_BAR"
```

The following example shows a utility subroutine that is passed the logical unit number, the window name, a list index, and the desired background color. String variables are used to refer to alternative lists of window attributes. The status of the FOPEN operation is returned in the variable error.

```
.PROGRAM open.window(lun, $name, type, color, error)

AUTO $list[2]

; Define sets of common attributes

$list[1] = "/MAXSIZE 100 100 /NOSCROLL_BAR /BUFFER"
$list[2] = "/MAXSIZE 300 200 /SCROLL_BAR /SHOW_SCROLL
      /BUFFER"

; Open the window

ATTACH (lun, 4) "GRAPHICS"
FOPEN (lun) $name, $list[type], "/BACKGROUND", color

error = IOSTAT(lun) ;Return error status

RETURN
```

Set up a TCP server with local port #260 to accept 5 client connections, with a buffer size of 1024 bytes:

```
FOPEN (lun, 16) "/LOCAL_PORT 260 /CLIENTS 5 /BUFFER_SIZE
1024"
```

Set up a TCP client connection that connects to port number 260 on the server called `server1`, and allocate a buffer size of 1024 bytes:

```
FOPEN (lun, 0) "server1 /REMOTE_PORT 260 /BUFFER_SIZE 1024"
```

Related Keywords

ATTACH	(program instruction)
DETACH	(program instruction)
FCLOSE	(program instruction)
FDELETE	(program instruction)
FEMPTY	(program instruction)
FOPEN_	(program instruction)
FSET	(program instruction)
IOSTAT	(real-valued function)
KILL	(monitor command and program instruction)

Syntax

```
FOPEN_ (lun, record_len, mode) file_spec
```

Function

Open a disk file for read-only, read-write, read-write-append, or read-directory, respectively, as indicated by the last letter of the instruction name.

Usage Considerations

The forms of FOPEN_ are:

- FOPENA
- FOPEND
- FOPENR
- FOPENW

See the Details section for descriptions of each instruction.

A logical unit must be attached before an open operation will succeed.

Each program task can have up to four disk files open at one time (one on each of the disk logical units). However, no more than 15 disk files can be open by the entire system at any time. That includes files opened by all of the program tasks and by the system monitor (for example, for an FCOPY command). (The error *Device not ready* results if an attempt is made to open a 16th file.)

Parameters

lun — Real-valued expression defining the logical unit number of the disk device to be accessed. (See the ATTACH instruction for a description of unit numbers.)

record_len — Optional real-valued expression defining the length of records to be read and written.

If the record length is omitted or is 0, variable-length records will be processed. In this case, random access of records cannot be done.

If the record length is nonzero, it specifies the length (in characters) of fixed-length records to be processed. Random access is allowed with fixed-length records.

mode — Optional real-valued expression defining how read access is to be done. The value specified is interpreted as a sequence of bit

flags as detailed below. (All bits are assumed to be clear if no mode value is specified.)

Bit 1 (LSB) Disable prereads (mask value = 1)

If this bit is clear, V+ will read a record as soon as the file is opened (a preread) and after each READ instruction in anticipation of subsequent READ requests. If this bit is set, no such prereads will be performed.

Bit 2 Enable random access (mask value = 2)

If this bit is clear, the file will be accessed sequentially. That is, records will be read or written in the order they occur in the file.

If this bit is set, the file will be accessed using random access (which is allowed only for disk files with fixed-length records). In random-access mode, the record-number parameter in the READ or WRITE instruction specifies which record will be accessed.

Bit 4 Force disk write (mask value = 8)

If set for a disk file being opened for write access, the physical disk will be written every time a record is written. In addition, the directory or file allocation information is updated with each write. This mode is equivalent to (but faster than) closing the file after every write. It is *much slower* than normal buffered mode, but it guarantees that information that is written will not be lost due to a system crash or power failure. This mode is intended primarily for use with log files that are left opened over an extended period of time and intermittently updated. For these types of files, the additional (significant) overhead of this mode is not as important as the benefit.

`file_spec` String constant, variable, or expression specifying the file to be opened. The string may contain an optional disk unit and an optional directory path, and must contain a file name, a period (.), and a file extension. (For FOPEND, the file name and extension are optional, and both can contain wildcard characters—see below.)

The current default disk unit and directory path are considered as appropriate (see the ***V+ Operating System User's Guide***).

The file specification may **not** include a physical device (for example, KERMIT>). If that component is needed, it must be specified when the logical unit is attached.

Details

This instruction opens a disk file so that input/output (I/O) operations can be performed. When the I/O operations are complete, the file should be closed using an FCLOSE or DETACH instruction.

FOPENA — opens a file for read-write-append access. If the specified file does not already exist, the file will be created.

If the file already exists, no error will occur, and the file position will be set to the end of the file. Then write operations will append to the existing file.

FOPEND — opens a disk directory for reading. The file name and extension in the file_spec parameter are used to prepare a file name template for use when read operations are later performed. Those read operations will return only records from the disk directory file that match the file name template. Any attempt to write to the directory file will cause an error. (See the ***V+ Language User's Guide*** for information on the format of directory records.)

The file name and extension can include wildcard characters (asterisks, *). A wildcard character *within* a file name or extension indicates that any character should be accepted in that position. A wildcard character at the *end* of a file name or extension indicates that any trailing characters are acceptable. A wildcard character in place of a file name (or extension) indicates that any name (or extension) is acceptable. Omission of the file name, the period, and the file extension is equivalent to specifying *.*. Omission of the period and file extension is equivalent to specifying a wildcard extension.

The FOPEND instruction cannot access the KERMIT device.

FOPENR — opens a file for read-only access. If the file does not already exist, an error will occur. Any attempt to write to the file will cause an error.

FOPENW — opens a file for read-write access. If the file already exists, an error will occur.

Any error in the specification of this instruction (such as attempting to access an invalid unit) will cause a program error and will halt program execution. However, errors associated with performing the actual operations (such as device not ready) do not halt program execution since these errors can occur in the normal operation of a program. These normal errors can be detected by using the IOSTAT function.

Example

```
FOPENR (5) "data.dat"
```

Open the file named data.dat on the default device for read-only access with variable-length records (record length omitted). Since the mode parameter is omitted, prereads will occur and the records will be accessed sequentially (which is required for variable-length records).

```
FOPENW (5, 32, 3) "A:x.d"
```

Open the file named x.d on the device A for read-write access using fixed-length records of 32 characters each. The mode value 3 has both bits 1 and 2 set; thus, prereads are to be disabled and random access is to be used.

```
FOPEND (5) "*.dat"
```

Open the current default directory to find all the files with the extension dat.

Related Keywords

ATTACH (program instruction)

DETACH (program instruction)

FCLOSE (program instruction)

FOPEN (program instruction)

IOSTAT (real-valued function)

KILL (monitor command and program instruction)

Syntax

```
FOR loop_var = initial TO final STEP increment
   group_of_steps
END
```

Function

Execute a group of program instructions a certain number of times.

Usage Considerations

An END instruction must be included in a program to match every FOR.

Parameters

loop_var	Real valued variable that is initialized when the FOR instruction is executed, and is incremented each time the loop is executed (cannot be a specified value or expression).
initial	Real value that determines the value of the loop variable the first time the loop is executed.
final	Real value that establishes the value to be compared to the loop variable to determine when the loop should be terminated.
increment	Optional real-value that establishes the value to be added to the loop variable every time the loop is executed. If omitted, the increment defaults to one, and the keyword STEP may also be omitted.

Details

The instructions between the FOR statement and the matching END statement are executed repeatedly, and **loop_var** is changed each time by the value of increment.

The processing of this structure is as follows:

1. When the FOR statement is first entered, set **loop_var** to the **initial** value.
2. Determine the values of the increment and **final** parameters.

3. Compare the value of **`final`** to the value of **`loop_var`**:
   a. If `increment` is positive and **`loop_var`** is greater than **`final`**, skip to item 7 below.
   b. If `increment` is negative and **`loop_var`** is less than (that is, more negative than) **`final`**, skip to item 7 below.
4. Execute the group of instructions following the FOR statement.
5. When the END step is reached, add the value of `increment` to the loop variable.
6. Go back to item 3 above.
7. Continue program execution at the first instruction after the END statement. **`loop_var`** retains the value it had at the time of the test in item 3 above.

Note that the group of instructions in the FOR structure may not be executed at all if the test in item 3 fails the first time.

The values of **`initial`**, `increment`, and **`final`** when the FOR statement is first executed determine how many times the group of instructions will be executed. Any changes to the values of these parameters within the FOR loop will have no effect on the processing of the FOR structure.

Changes to the loop variable within the loop will affect the operation of the loop and should normally not be done.

> **NOTE:** If **`initial`**, **`final`**, or `increment` are not integer values, rounding in the floating point computations may cause the loop to be executed more or fewer timer than expected.

Example

The following example sets all elements of a 10x10 array to 0:

```
FOR i = 1 TO 10
   FOR j = 1 TO 10
     array[i,j] = 0
   END
END
```

Related Keywords

DO ... UNTIL (program instructions)

EXIT (program instruction)

NEXT (program instruction)

WHILE ... DO (program instruction)

Syntax

FORCE._

Function

AdeptForce option status and control instructions.

Usage Considerations

The forms of FORCE._ are:

FORCE.FRAME	Set transformation for force reference frame
FORCE.MODE	Set and control force operating modes
FORCE.OFFSET	Set temporary or permanent force offset
FORCE.READ	Return current force reading

Details

These instructions are part of the AdeptForce VME option.

See the *AdeptForce VME User's Guide* for full syntax and details.

Related Keywords

SELECT	(real-valued function)
LATCH	(transformation function)
LATCHED	(real-valued function)
#PLATCH	(precision-point function)

Syntax

FRACT (value)

Function

Return the fractional part of the argument.

Parameter

value Real-valued expression whose fractional part is returned by this function.

Details

The fractional part of a real value is the portion to the right of the decimal point (when the value is written without the use of scientific notation).

The value returned has the same sign as the function argument.

Examples

```
FRACT(0.123)     ;Returns 0.123
FRACT(-5.462)    ;Returns -0.462
FRACT(1.3125E+2) ;Returns 0.25 (1.3125E+2 = 131.25)
```

Related Keyword

INT (real-valued function)

Syntax

```
FRAME (location_1, location_2, location_3, location_4)
```

Function

Return a transformation value defined by four positions.

Parameters

location_1 Transformation, compound transformation, or a transformation-valued function whose position is used to define the X axis of the computed frame.

location_2 Transformation, compound transformation, or a transformation-valued function whose position is used to define the X axis of the computed frame.

location_3 Transformation, compound transformation, or a transformation-valued function whose position is used to define the Y axis of the computed frame.

location_4 Transformation, compound transformation, or a transformation-valued function whose position is returned as the position of the computed frame transformation.

Details

While the robot can be used to define an X, Y, Z position very accurately, it is often difficult to define precisely an orientation. For applications such as palletizing, the FRAME function is very useful for accurately defining a base transformation whose position and orientation are determined by four positions. This function returns a transformation value that is computed as follows:

1. Its origin is at the point defined by **location_4**.
2. Its positive X axis is parallel to the line passing through the points defined by **location_1** and **location_2**, in the direction from **location_1** to **location_2**.
3. Its X-Y plane is parallel to the plane that contains the points defined by **location_1**, **location_2**, and **location_3**.
4. Its positive Y direction is from the computed X axes (as defined above), toward the point defined by **location_3**.

Example

The following instruction defines the transformation base.frame to have the same X, Y, Z position as origin, its X axis parallel to the line from center to x, and its Y axis approximately in the same direction as the line from center to y.

```
SET base.frame = FRAME(center, x, y, origin)
```

Related Keyword

TRANS (transformation function)

Syntax

```
FREE (memory, select)
```

Function

Return the amount of unused free memory storage space.

Parameters

`memory` Optional real value, variable, or expression (interpreted as an integer) that specifies which portion of system memory is to be examined, as shown below. The value 0 is assumed if the parameter is omitted.

memory	Memory examined
0	Program memory
1	Graphics memory
2	Vision memory

`select` Optional real value, variable, or expression (interpreted as an integer) that specifies what information about the memory is to be returned, as shown below. The value 0 is assumed if the parameter is omitted.

NOTE: If both parameters are omitted, the parentheses must still be included.

Details

This function returns the information displayed by the FREE command. Unlike the FREE command, however, this function returns only one value, as requested with the parameters.

As shown above, the `memory` parameter specifies which portion of system memory is to be examined. The `select` parameter specifies which item of information is to be returned for that portion of memory, as follows:

select	Information returned
0	Percentage of memory available
1	Available memory, in kilobytes (1024 bytes)
2	Number of segments in program memory (memory = 0)
	Number of free windows (memory = 1)
	% of vision memory used for models (memory = 2)

If the controller does not have a graphics system processor, this function always returns zero when the memory parameter is 1. If the controller does not include the optional AdeptVision system, the function always returns zero when the memory parameter is 2.

Related Keyword

FREE	(monitor command, see the *V+ Operating System Reference Guide)*

Syntax

```
FSEEK (logical_unit, record_number)
```

Function

Position a file open for random access and initiate a read operation on the specified record.

Usage Considerations

A file must be open for random access on the specified logical unit (see the FOPEN_ instruction).

For efficiency in most applications, the file should be opened in no preread mode.

Parameters

`logical_unit` Real-valued expression that identifies the device to be accessed. (See the ATTACH instruction for a description of unit numbers.)

`record_number` Optional real-valued expression that specifies the record to read for file-oriented devices opened in random-access mode. If omitted, the record following the one last read is assumed.

Details

When a file is open for random access, system performance can be improved by overlapping the time required for disk file access with processing of the current data. By using the FSEEK instruction, an application program can initiate a disk seek and possible read operation immediately after a READ instruction is processed but before processing the data.

Any error in the specification of this instruction (such as referencing an invalid unit) will cause a program error and will halt program execution. However, errors associated with performing the actual seek operation (such as end of file or device not ready) will not halt program execution since these errors may occur in the normal operation of a program. These normal errors can be detected by using the IOSTAT function after performing the subsequent READ operation. In general, it is good practice always to test whether each file operation completed successfully by testing the value from IOSTAT.

Example

```
Seek record number 130 in the file open on logical unit 5:
FSEEK (5, 130)
```

Related Keywords

ATTACH (program instruction)

FOPEN_ (program instruction)

IOSTAT (real-valued function)

READ (program instruction)

Syntax

```
FSET (logical_unit) attribute_list
```

Function

Set or modify attributes of a graphics window, serial line, or network device related to AdeptNet.

Usage Considerations

If a window has been referenced, it must have been opened already with an FOPEN instruction. If a serial line is referenced, it must have been attached already with an ATTACH instruction.

The use of this instruction with NFS or TCP network devices applies only to systems fitted with the AdeptNet option and with the appropriate license(s).

Parameters

`logical_unit` Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)

`attribute_list` List of string constants, variables, and expressions; real values, variables, and expressions; and format specifiers used to define the characteristics of the window. See the description of the FOPEN instruction for detailed information on this parameter.

Details

Using FSET with Windows

This instruction sets attributes for a window that already exists and is already open for access.

> **NOTE:** No interlocks exist to prevent multiple program tasks from changing the attributes of a window.

The argument list for this instruction has exactly the same format as that for the FOPEN instruction, except that no window name is included for this instruction. Refer to the description of the FOPEN instruction for information on how window attributes are specified.

The window attributes described below can be specified with either the FSET or FOPEN instructions (unless noted otherwise). For some attributes, after the attribute has been specified, its effect may be modified later in one of the following ways:

- The following keywords set attributes that can be changed only by another FSET instruction.

/BORDER	/LUT	/SPECIAL
/EVENT	/MARGINS	/TERMINAL
/FONT	/MENU	/UPDATE
/FONT_HDR	/CURSOR	/V_ARROWINC
/H_ARROWINC	/NOUPDATE	/V_RANGE
/H_RANGE	/POINTER	

- The following keywords set attributes that can be changed by another FSET instruction or by pointer device activity.

/ABS_POSITION	/NODISPLAY	/SIZE
/DISPLAY	/POSITION	/STACK
/H_HANDLE	/SELECT	/V_HANDLE
/H_SCROLL	/SHOW	/V_SCROLL

- The following keywords are related to the pull-down menu and do not actually remain as attributes of the window.

/IGNORE	/PULLDOWN

This information is provided for each attribute described in **Table 2-9**:

- Keyword that identifies the attribute

  Some attributes can be either on or off. For each such attribute there are two, corresponding keywords. The on keyword directly relates to the attribute (for example, /DISPLAY); the off keyword has a NO prefix (for example, /NODISPLAY).

- Description of the effect of the attribute

  For attributes with an on/off character, the on condition is described.

- Description of any arguments required for the attribute
- Default attribute setting

Table 2-9. FSET Graphics Window Attributes

Attribute:	**/ABS_POSITION**
Explanation:	Position the window at an absolute location on the screen rather than at a location relative to the parent window.
Arguments:	The X and Y offsets from the upper left corner of the monitor.
Default:	None
Attribute:	**/BORDER**
Explanation:	Enable a border around all or part of the window. The border is a line (one pixel wide) inside the window boundary, and it will obscure anything in that area. The border is always drawn with color 1 (black).
Arguments:	One or more of the following keywords. • ALL = Display all borders • NONE= Display no borders • TOP = Display top border • NOTOP = Suppress top border • BOTTOM = Display bottom border • NOBOTTOM = Suppress bottom border • LEFT = Display left border • NOLEFT = Suppress left border • RIGHT = Display right border • NORIGHT = Suppress right border
Default:	/BORDER ALL
Attribute:	**/CURSOR**
Explanation:	Set the mouse cursor to a loaded icon.

Table 2-9. FSET Graphics Window Attributes (Continued)

Arguments:	The index of the icon element to use, followed by the name of the icon (in single quotes)
Default:	None
Attribute:	**/DISPLAY and /NODISPLAY**
Explanation:	Display the window. Top-level windows that are not displayed may be displayed by selecting their name from the Adept pull-down menu at the top of the screen.
Arguments:	None
Default:	/DISPLAY
Attribute:	**/EVENT**
Explanation:	Enables or disables event processing for the window. If you want to receive events, you must explicitly use this attribute with FSET or FOPEN each time you open a window. The details of event processing, and descriptions of the events that can be specified with this keyword, are described with the GETEVENT instruction. If the window is opened for write-only access (see the /WRITEONLY attribute), this keyword causes the error File already opened.
Arguments:	One or more keywords (see GETEVENT)
Default:	/EVENT NONE
Attribute:	**/EVENT CONNECT**
Explanation:	Enables or disables connect and disconnect event notification. If the event value returned is 20, then you are connected to the PC. If the event value returned is 21, then you are disconnected from the PC.
Arguments:	None
Default:	None

Table 2-9. FSET Graphics Window Attributes (Continued)

Attribute:	**/FONT**
Explanation:	Selects a character font to be used for all text output in the window, except for output from the GTYPE instruction. The new font will take effect immediately, with the following effect on the placement of the next character output to the window: (1) The new character cell will have its left edge at the same pixel column as before the font change. (2) Its baseline will remain at the same pixel row. (3) If the top of the character cell extends above the text top margin, the character position will be bumped down so that no part of the cell is outside the text margins. (4) A similar rule applies if the right edge of the character cell extends beyond the text right margin. Font number 1 is the standard font—it has character cells 8 pixels wide and 15 pixels high; there are 3 pixels below the baseline; and uppercase letters are 9 pixels high.
Arguments:	Font number (the value must be greater than or equal to one)
Default:	/FONT 1
Attribute:	**/FONT_HDR**
Explanation:	Specify the font number to use for the title, menu bars, and pull-down menus of the window. (Changes the title bar immediately, changes the menu bars at the next FSET /MENU... instruction.)
Arguments:	Font number
Default:	/FONT 1
Attribute:	**/H_ARROWINC**
Explanation:	Sets the change in the position of the scroll handle in the horizontal scroll bar that will be caused by clicking on the arrow buttons in the horizontal scroll bar. (This keyword is valid only when the event GRAB_H_SCROLL is enabled [see the GETEVENT instruction].)
Arguments:	Number of units (see /H_RANGE) to move the handle for each click on an arrow button (the value must be greater than or equal to zero)
Default:	Pixel width of the current font cell
Attribute:	**/H_HANDLE**
Explanation:	Sets the displayed position of the scroll handle in the horizontal scroll bar. (This keyword is valid only when the event GRAB_H_SCROLL is enabled [see GETEVENT].)
Arguments:	Relative position along bar (note that this is **not** a pixel count; 0 = left-most end, right-most end may be set with /H_RANGE)

Table 2-9. FSET Graphics Window Attributes (Continued)

Default:	Current scroll position
Attribute:	**/H_RANGE**
Explanation:	Sets the value of the right-most position of the scroll handle in the horizontal scroll bar. (This keyword is valid only when the event GRAB_H_SCROLL is enabled [see GETEVENT].)
Arguments:	Value associated with the right-most position of the scroll handle (value must be greater than or equal to 0; if n handle positions are desired, use the value n–1)
Default:	Total number of pixels that the window can be scrolled (**not** the width of the window)
Attribute:	**/H_SCROLL**
Explanation:	Sets the horizontal scroll position of the displayed contents of the window. (This keyword is ignored if it is specified with FOPEN and the window already exists.)
Arguments:	Horizontal offset (in pixels) in the bitmap to the first column to display (the value must be greater than or equal to zero)
Default:	/H_SCROLL 0
Attribute:	**/IGNORE**
Explanation:	Specifies pull-down menu items that cannot be selected. These items are dimmed in the pull-down menu display. This keyword must be specified in the same FSET instruction as the /PULLDOWN keyword.
Arguments:	List of numbers in the range 1 to 30, which correspond to rows in the pull-down menu. Numbers greater than the number of pull-down items are ignored.
Default:	No pull-down items are ignored
Attribute:	**/LUT**
Explanation:	Sets the red, green, and blue (RGB) values of a color. The specified color is actually altered for the entire graphics system. Thus, this keyword must be used with care. Note, for example, that changing color values may cause all the color descriptions in this manual to become invalid.
Arguments:	The color to be changed (in the range 0 to 15), followed by the red, green, and blue values for the color (each in the range 0 to 255). A value of –1 may be specified for the red, green, or blue value to return that value to its default. For instance, specifying /LUT 15 127 127 127 will change the default white (color #15) in the system to a gray, and specifying /LUT 15 –1 –1 –1 will return the white to its default color.

Table 2-9. FSET Graphics Window Attributes (Continued)

Default:	System default colors (see GCOLOR)
Attribute:	**/MARGINS**
Explanation:	Sets the text margins for the window. This keyword also sets the text cursor position to the top left corner of the text window defined by the new margins. The text scrolling window is reset to its maximum size.
Arguments:	Coordinates of the left, top, right, and bottom edges of the text window (if a value is negative, the corresponding margin is not modified)
Default:	• X coordinate of left margin = 8 • Y coordinate of top margin = 8 • X coordinate of right margin = (right_edge_of_window – 8) • Y coordinate of bottom margin = (left_edge_of_window – 8)
Attribute:	**/MENU**
Explanation:	Specifies up to 20 items to appear in the menu bar of the window. Each menu string is displayed in a box in the menu bar, left to right in the order specified. A space is added to each side of each string. If the strings are too long, the excess fall off the end of the menu bar without any error indication. The strings are written in the current font defined for the menu bar. (This keyword is ignored if the window does not have a menu bar.)
Arguments:	List of up to 20 strings, each one enclosed in single quotes. Each string delimited by single-quote characters must be separated from the next string by at least one space. The correct way to clear the menu bar is to specify no strings, not a null string (that is, with /MENU, not with /MENU ' ').
Default:	/MENU (no items)
Attribute:	**/POINTER**
Explanation:	Specifies the point of reference for position information obtained from the mouse cursor. This keyword takes effect with the next event that occurs—it does not change any events that have already occurred but are waiting to be read.

Table 2-9. FSET Graphics Window Attributes (Continued)

Arguments:	• DISPLAY = Return pointer device positions relative to the upper left-hand corner of the window region displayed on the screen. • SCREEN = Return pointer device positions relative to the upper left-hand corner of the entire screen. • WINDOW = Return pointer device positions relative to the upper left-hand corner of the allocated window, even if the corner is scrolled out of the displayed region.
Default:	/POINTER WINDOW
Attribute:	**/POSITION**
Explanation:	Sets the displayed position of the window relative to its parent. The arguments define the position of this window's upper left-hand corner relative to the upper left-hand corner of the parent's allocated window. If this is a top-level window, the position is relative to the upper left-hand corner of the screen. (This keyword may not be specified for control-bar windows.) This keyword is ignored if it is specified with FOPEN and the window already exists. If this keyword is used with FSET, the value specified overrides any mouse drag that may have been done by the user. This keyword may also specify the position of the reference point of an icon relative to the top left corner of the icon (see the /ICON keyword for FOPEN).
Arguments:	X position (in pixels) relative to parent (always rounded up to be an even number) Y position (in pixels) relative to parent
Default:	/POSITION 0 0

Table 2-9. FSET Graphics Window Attributes (Continued)

Attribute:	**/PULLDOWN**
Explanation:	Outputs text into a pull-down window, and displays the pull-down window under the specified menu-bar item if that item is currently selected (highlighted). (This keyword may not be specified with FOPEN.) All the strings specified are displayed, left justified, stacked vertically, in a window the width of the longest string (plus a space on each side of the string). With the standard font, the pull-down window may have a maximum area equivalent to 25 lines of text, 25 characters wide. There can be a maximum of 30 lines of text. If there are more than 25 lines, the maximum pull-down will be less than 25 characters wide; conversely, if there are fewer than 25 lines, the pull-down can be more than 25 characters wide. Each line of text will be truncated on the right if necessary. (See the /IGNORE keyword for information on how to disable pull-down items.) Only one pull-down window can be present in the entire system at any one time.
Arguments:	Number of the menu-bar item with which this pull-down list is associated (1 [left-most item] to N [right-most item]) followed by a list of strings (each in single quotes) separated by spaces
Default:	No pull-down menu specified
Attribute:	**/SELECT**
Explanation:	Specifies that a window is selected for receiving input. The effect of this keyword is equivalent to clicking the mouse on the window.
Arguments:	None
Default:	The window is not selected
Attribute:	**/SHOW**
Explanation:	Forces a window and all its ancestors to be displayed. (This keyword differs from /DISPLAY—that keyword does not display a window if any one of its ancestors is not displayed.)
Arguments:	None
Default:	None

Table 2-9. FSET Graphics Window Attributes (Continued)

Attribute:	**/SIZE**
Explanation:	Sets the size of the window in the display. The size specified overrides any mouse drag that may have been done by the user. This keyword may not be specified for control-bar windows. (This keyword is ignored if it is specified with FOPEN and the window already exists.) The X value is always rounded up to be an even number. For windows with a title bar, X and Y values less than 64 are set to 64. For windows without a title bar, X and Y values less than 8 are set to 8. The values of X and Y are clipped to the full size of the window as required. This keyword may also specify the size of an icon or font (see the /ICON or /DEFFONT keyword for the FOPEN instruction).
Arguments:	• X size (in pixels) • Y size (in pixels)
Default:	Display the whole window (see /MAXSIZE described in the FOPEN instruction)
Attribute:	**/SPECIAL**
Explanation:	Sets special attributes for the window.
Arguments:	One or more of the following keywords. • DESELECT = Allow deselection • NODESELECT = Don't allow deselection • POSITION = Allow moving by dragging title bar • NOPOSITION = Don't allow dragging of title bar • SELECTABLE = Allow window to be selected from pull-down menu under Adept icon • NOSELECTABLE = Don't allow window to be selected from pull-down menu under Adept icon. (Window name is dimmed in pull-down menu.) • SIZE = Allow resize with sizing icon • NOSIZE = Don't allow resize with sizing icon
Default:	/SPECIAL DESELECT POSITION SELECTABLE SIZE

Table 2-9. FSET Graphics Window Attributes (Continued)

Attribute:	**/STACK**
Explanation:	Sets the position of the window in its window stack, which determines the layering of the windows in the display. This keyword may not be specified for control-bar windows. No child of a window can be stacked below that window's control bars.
Arguments:	Code value for desired position: • 1 Move to top of parent's stack • –1 Move to bottom of parent's stack
Default:	/STACK 1
Attribute:	**/TERMINAL**
Explanation:	Sets characteristics of the terminal emulator used to output text to the window.
Arguments:	One or more of the following keywords. • CURSOR = Display the text cursor • NOCURSOR = Do not display text cursor • OVERSTRIKE = Force designated overstrike characters to overstrike the previous character • NOOVERSTRIKE = defeat overstriking • WRAP = Wrap lines of text that are too long • NOWRAP = Truncate long lines of text
Default:	/TERMINAL CURSOR OVERSTRIKE WRAP
Attribute:	**/TITLE**
Explanation:	Set the window title, which is the text that is displayed in the title bar. The title has no other meaning to the V+ system.
Arguments:	String enclosed in single quotes. The string will be truncated to 40 characters.
Default:	Same as the window name

Table 2-9. FSET Graphics Window Attributes (Continued)

Attribute:	**/UPDATE and /NOUPDATE**
Explanation:	In normal operation, the screen display is updated every time new graphics appear in any displayed window. The /NOUPDATE keyword stops this process for the window currently open, thereby hiding all new graphics from the screen display until normal operation resumes. The /UPDATE keyword will force the screen display to be updated and normal operation to resume. (This keyword may not be specified with the FOPEN instruction.)
Arguments:	None
Default:	/UPDATE
Attribute:	**/V_ARROWINC**
Explanation:	Sets the change in the position of the scroll handle in the vertical scroll bar that will be caused by clicking on the arrow buttons in the vertical scroll bar. (This keyword is valid only when the event GRAB_V_SCROLL is enabled [see the GETEVENT instruction].)
Arguments:	Number of units (see V_RANGE) to move the handle for each click on an arrow button (the value must be greater than or equal to zero)
Default:	Pixel height of the current font cell
Attribute:	/V_HANDLE
Explanation:	Sets the displayed position of the scroll handle in the vertical scroll bar. (This keyword is valid only when the event GRAB_V_SCROLL is enabled [see GETEVENT].)
Arguments:	Relative position along bar (note that this is **not** a pixel count; 0 = top-most end; bottom-most end may be set with /V_RANGE)
Default:	Current scroll position
Attribute:	**/V_RANGE**
Explanation:	Sets the value of the bottom-most position of the scroll handle in the vertical scroll bar. (This keyword is valid only when the event GRAB_V_SCROLL event is enabled [see GETEVENT].)
Arguments:	Value associated with bottom-most position of the scroll handle (the value must be greater than or equal to 0; if n handle positions are desired, use the value n–1)
Default:	Total number of pixels that the window can be scrolled (not the height of the window)

Table 2-9. FSET Graphics Window Attributes (Continued)

Attribute:	/V_SCROLL
Explanation:	Sets the vertical scroll position of the displayed contents of the window. (This keyword is ignored if it is specified with FOPEN and the window already exists.)
Arguments:	Vertical offset (in pixels) in the bitmap to the first row to display (the value must be greater than or equal to zero)
Default:	/V_SCROLL 0

Using FSET with Serial Lines

The following specifications can be used as arguments to directly ATTACH a serial line:

LOCAL.SERIAL:n Local serial line n on the local CPU. For Adept CPU boards, n = 1 or 2.

SERIAL:n Global serial line n on the Adept SIO board. For Adept SIO board, n = 1, 2, 3, 4 (4 cannot be used if MCP is installed).

As a convenience, the following synonyms may be used:

KERMIT The serial line currently configured for Kermit protocol.

MONITOR The serial line currently configured for use as the monitor terminal.

The keywords listed in **Table 2-10** may appear in the keyword list string.

Table 2-10. FSET Serial Line Attributes

Attribute	Argument	Description
/PARITY	NONE	No parity generation
	EVEN	Use even parity
	ODD	Use odd parity
/STOP_BITS	1 or 2	Use 1 or 2 stop bits per byte
/BYTE_LENGTH	7 or 8	Use 7 or 8 bits per byte

Table 2-10. FSET Serial Line Attributes (Continued)

Attribute	Argument	Description
/FLOW	NONE	No flow control
	XON_XOFF	Detect and generate XON/XOFF (turn off modem)
	MODEM	Use modem control RTS/CTS (turn off XON_XOFF)
/DTR	OFF	Turn off the DTR modem signal
	ON	Turn on the DTR modem signal
/MULTIDROP	OFF	Do not use multi-drop mode
	ON	Use multidrop mode (Only valid for LOCAL.SERIAL:1 on Adept CPUs)
/FLUSH	OFF	Disable recognition of Ctrl+O for flushing output
	ON	Enable recognition of Ctrl+O for flushing output
/SPEED	110, 300, 600, 1200, 2400, 4800, 7200, 9600, 19200, 38400	Select the indicated baud rate. For current Adept boards, a baud rate of 19200 is incompatible with a baud rate of 7200 or 38400 on a pair-wise basis. The pairs are: (LOCAL.SERIAL:1, LOCAL.SERIAL:2) (SERIAL:1, SERIAL:4) (SERIAL:2, SERIAL:3)

Drivers for KERMIT, DDCMP, and NETWORK do not support all modes indicated by the keywords; they ignore those that are not supported.

Using FSET with NFS and TCP

The following AdeptNet devices may be referenced with the FSET instruction:

NFS Network File System

TCP Transmission Control Protocol

You can use the attributes listed in **Table 2-11** when accessing these devices with the FSET instruction.

Table 2-11. FSET Attributes for AdeptNet

Attribute	Description
/ADDRESS	IP address. (Applies only to the TCP device.)
/MOUNT	Defines the name to be used for an NFS server remote disk. (Applies only to the NFS device.)
/NODE	Node name.
/PATH	Path for NFS server remote disk. (Applies only to the NFS device.)

You may define new nodes on the network using the FSET program instruction to access a logical unit that has been attached to the TCP device. The string used with the FSET instruction has the same format as that used with the NODE statement in the V+ configuration file (see the later example).

You may also define new remote mounts, using the FSET program instruction to access a logical unit that has been attached to the NFS device. The string used with the FSET instruction has the same format as that used with the MOUNT statement in the V+ configuration file (see the later example).

Examples

Graphics

The following FSET instruction causes a pull-down menu to be displayed under the menu-bar item just selected by the user (assuming that the array element event[2] has been set by a GETEVENT instruction that detected a menu event):

```
FSET (lun) "/PULLDOWN", event[2], $menu[event[2]]
```

This instruction requires that string array elements $menu[n] be defined for each of the menu items displayed on the menu bar. The following line shows how a pull-down list with four items could be defined for the first item on the menu bar. (Note that the menu items can have different lengths, they can consist of multiple words, and there is a space between the single quotes delineating items.)

```
$menu[1] = "'Item #1' 'Second Item' 'One more item' 'Last
item'"
```

The following instruction takes control of the vertical scroll bar away from the window manager. The vertical scroll bar is set to return values from 0 to 10. The initial position of the scroll-bar handle is at the top (zero) and each time the up-arrow or down-arrow icon is clicked, the value of the scroll-bar handle position changes by 2 units.

```
FSET (lun) "EVENT GRAB_V_SCROLL/V_RANGE 10/V_HANDLE
0/V_ARROWINC 2"
```

The following example attaches to serial line 2 and sets the baud rate to 2400:

```
ATTACH (slun, 4) SERIAL:2
FSET (slun) "SPEED 2400"
```

AdeptNet

Define a new node called SERVER2 with the IP address 192.168.144.102:

```
ATTACH (lun, 4) "TCP"
FSET (lun) "/NODE 'SERVER2' /ADDRESS 192 168 144 102"
```

Define a new NFS mount with the disk name DISK2 to access the exported directory `/a` of the node (server) called SERVER1:

```
ATTACH (lun, 4) "NFS"
FSET (lun) "/MOUNT 'DISK2' /NODE 'SERVER1' /PATH '/a'"
```

Related Keywords

FOPEN (program instruction)

IOSTAT (real-valued function)

Syntax

```
GAIN.SET set, motor
```

Function

Select a set of feedback gain parameters for one or more motors of the currently selected robot.

Usage Considerations

This instruction currently applies only to motor 4 of the AdeptOne-MV and AdeptThree-MV robots. For the other motors of these robots and for all other robots, this instruction has no effect (and causes no error).

This instruction affects the robot currently selected with a SELECT program instruction. The GAIN.SET instruction takes effect immediately and is not synchronized with robot motion segments. A task does not have to be attached to the robot to issue this instruction.

Parameters

`set`	Optional real expression that specifies the gain set to use. The default set is selected if this parameter is zero or omitted.
`motor`	Optional real expression that specifies the motor of the selected robot. If this parameter is zero or omitted, the gain set for all motors of the selected robot is selected. Otherwise, only the specified motor is affected.

Details

The gain set must be one of several that are predefined.

This instruction changes some servo-tuning values, specifically those for feedforward gain, for certain axes. You can select the most suitable values depending on payload mass, payload inertia, and application requirements. If necessary, you can use this instruction every time the payload changes (for example, before and after the robot picks up a heavy object). Your robot instruction handbook may have additional information specific to your robot.

You can use the PAYLOAD instruction to adjust feedforward gains in a predetermined way, just as you can use the GAIN.SET instruction to adjust feedback gains in a predetermined way. Because GAIN.SET does not affect the feedforward, there is no conflict between these two instructions. That is, the order of their execution is unimportant.

Related Keywords

ACCEL	(program instruction)
PAYLOAD	(program instruction)
SPEED	(monitor command and program instruction)

Syntax

```
GARC (lun, mode) xc, yc, radius, ang0, angn
```

Function

Draw an arc or a circle in a graphics window.

Usage Considerations

This instruction is available only with an graphics-based system.

Parameters

lun — Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)

mode — Optional real value, variable, or expression (interpreted as a bit field) specifying how the instruction is to operate. Zero is assumed if the parameter is omitted. The bit interpretations are given below.

Bit 1 (LSB)Empty (0) vs. Filled (1)Mask value = 1

If this bit is set, the arc or circle is filled with the current foreground color. (Not implemented.)

Bit 2Use (0) vs. Ignore (1) anglesMask value = 2

If this bit is set, a full circle is drawn regardless of the values of the ang0 and angn parameters.

xc, yc — Real values, variables, or expressions (interpreted as integers) specifying the coordinates (in pixels) of the center of the circular arc. The origin for the coordinate system (0,0) is the top left corner of the window, with positive X directed to the right and positive Y directed toward the bottom.

radius — Real value, variable, or expression (interpreted as an integer) specifying the radius (in pixels) of the circular arc. If the value is zero, a point is drawn. Nothing is drawn if the value is negative.

`ang0, angn` Optional real values, variables, or expressions specifying starting and stopping angles (in degrees) for the arc. A circle is drawn if both parameters are omitted or both parameters have the same value. The angles are measured counterclockwise from a line pointing to the right.

Details

The arc or circle is drawn with the current graphics foreground color (see GCOLOR), texture (see GTEXTURE), and logical operation (see GLOGICAL). Opaque textures are always used, regardless of the current opaque/transparent mode setting (see GTEXTURE).

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

Examples

In the window open on graphics logical unit 20, draw an arc centered at coordinates (10,15) with 50-pixel radius, starting at angle 90 (top of circle) and moving counterclockwise to angle 0 (right side of circle):

```
GARC (20) 10, 15, 50, 90, 0
```

In the window open on graphics logical unit 20, draw a complete circle centered at coordinates (10,15) with 50-pixel radius (ignore the angle arguments):

```
GARC (20,2) 10, 15, 50, 90, 0
```

In the window open on graphics logical unit 20, draw a complete circle centered at coordinates (10,15) with 50-pixel radius:

```
GARC (20) 10, 15, 50
```

Related Keywords

FOPEN (program instruction)
GCLIP (program instruction)
GCOLOR (program instruction)
GLOGICAL (program instruction)
GTEXTURE (program instruction)
GTRANS (program instruction)

Syntax

GCHAIN (lun) x, y, **points, direction[**index**]**

Function

Draw a chain of points in a graphics window to form a complex figure.

Usage Considerations

This instruction is available only with an graphics-based system.

Parameters

lun — Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)

x, y — Optional real values, variables, or expressions (interpreted as integers) specifying the starting point for the chain. If both parameters are omitted, the chain is continued from the last chain point drawn since the last FOPEN instruction (or from [0,0] if there has been no previous GCHAIN since the last FOPEN). If either value is omitted, the other value must also be omitted.

points — Real value, variable, or expression (interpreted as an integer) specifying the number of points in the chain, *not* counting the starting point. The starting point is drawn, except when x and y are omitted. The acceptable range of **points** is 0 to 1008, inclusive.

direction[] — Real array that contains direction codes for the respective points in the chain. The values indicate the directions to the successive pixels in the chain, as follows (angles are measured counterclockwise from a line pointing to the right):

value	Direction	value	Direction
0	0 degrees	4	180 degrees
1	45 degrees	5	225 degrees
2	90 degrees	6	270 degrees
3	135 degrees	7	315 degrees

`index`	Optional real value, variable, or expression (interpreted as an integer) specifying the *first* array element to be accessed. Element zero is accessed first if no index is specified.

Details

This instruction draws a chain of pixels to form an arbitrary shape. The chain is drawn with the current graphics foreground color (see GCOLOR), texture (see GTEXTURE), and logical operation (see GLOGICAL). Opaque textures are always used, regardless of the opaque/transparent mode setting (see GTEXTURE).

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

Example

In the window open on graphics logical unit 20, draw a chain of 30 pixels starting at location (25,10). The relative position of each pixel is found in elements dir[0] through dir[29]:

```
GCHAIN (20) 25, 10, 30, dir[]
```

In the window open on graphics logical unit 20, draw a chain of 25 pixels, starting where the previous chain left off. The relative position of each pixel is in array elements dir[6] through dir[30]:

```
GCHAIN (20) , , 25, dir[6]
```

Related Keywords

FOPEN	(program instruction)
GCLIP	(program instruction)
GCOLOR	(program instruction)
GLOGICAL	(program instruction)
GPOINT	(program instruction)
GTEXTURE	(program instruction)
GTRANS	(program instruction)

Syntax

```
GCLEAR (lun)
```

Function

Clear an entire graphics window to the background color.

Usage Considerations

This instruction is available only with an graphics-based system.

Parameter

lun	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)

Details

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

This instruction also deletes all slide bars (see GSLIDE) in the window that is currently open.

Example

Set the entire window open on graphics logical unit 20 to the background color, erasing any current display and deleting any slide bars in that window:

```
GCLEAR (20)
```

Related Keywords

FOPEN	(program instruction)
GCOLOR	(program instruction)

Syntax

GCLIP (lun) x, y, dx, dy

Function

Set the clipping rectangle for all graphics instructions (except GFLOOD), to suppress all subsequent graphics that fall outside the rectangle.

Usage Considerations

This instruction is available only with an graphics-based system.

If the rectangle is set improperly, all subsequent graphics may be suppressed.

The clipping rectangle is canceled for the specified logical unit when a window is opened on the logical unit with the FOPEN instruction.

Parameters

lun	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
x, y	Optional real values, variables, or expressions (interpreted as integers) specifying the (pixel) coordinates of the top left corner of the clipping rectangle. Omitted parameters default to zero.
dx, dy	Optional real values, variables, or expressions (interpreted as integers) specifying the (pixel) width and height, respectively, of the clipping rectangle. Zero is assumed for an omitted parameter, unless all parameters are omitted or zero—see below.

Details

This instruction defines a clipping rectangle that limits the portion of the window affected by all the V+ graphics instructions, except for the GFLOOD instruction. That is, the graphics manager automatically suppresses any graphics output that would fall outside the clipping rectangle.

If all parameters are omitted or specified to be zero, the clipping rectangle is set to be the entire window. If the clipping rectangle is specified to extend beyond the window boundary, the rectangle will be limited to the window boundary.

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

Example

The following instruction sets the clipping rectangle (for the current window open on graphics logical unit 20) to have its top left-hand corner at location (10,15). The width is 100 pixels, and the height is 200 pixels. All subsequent graphics output for the logical unit that falls outside this rectangle will be suppressed:

```
GCLIP (20) 10, 15, 100, 200
```

Related Keywords

FOPEN (program instruction)

GARC (program instruction)

GCHAIN (program instruction)

GCOPY (program instruction)

GICON (program instruction)

GLINE (program instruction)

GLINES (program instruction)

GPANEL (program instruction)

GPOINT (program instruction)

GRECTANGLE (program instruction)

GSCAN (program instruction)

GSLIDE (program instruction)

GTYPE (program instruction)

Syntax

```
GCOLOR (lun) foregrnd, backgrnd
```

Function

Set the foreground and background colors for subsequent graphics output.

Usage Considerations

This instruction is available only with an graphics-based system.

The foreground and background colors are reset for the specified logical unit when a window is opened on the logical unit with FOPEN.

Parameters

lun	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
foregrnd	Optional real value, variable, or expression (interpreted as an integer) specifying the foreground color. Color values range from 0 to 15. The color is not changed if the parameter is omitted or if it has the value –1.
backgrnd	Optional real value, variable, or expression (interpreted as an integer) specifying the background color. Color values range from 0 to 15. The color is not changed if the parameter is omitted or if it has the value –1.

Details

The GCOLOR instruction sets the foreground and background colors for subsequent graphics output from any of the following instructions:

GARC	GLINE	GRECTANGLE
GCHAIN	GLINES	GSCAN
GCLEAR	GPANEL	GTYPE
GFLOOD	GPOINT	

For 16-color windows, the full range of color values (0 to 15) specify different colors. For 2-color windows, all even values (including 0) specify light blue, and all odd values specify black.

The standard color values are shown in Table 2-12. These colors may be changed by using the /LUT attribute for the FSET instruction.

This instruction operates on the window that is currently open for the specified logical unit. The effect of this instruction will apply to all subsequent output to the window. (Note, however, that if the window is open in buffered mode, the effect will not be displayed until the buffer fills or an FEMPTY instruction is executed.)

Table 2-12. Standard Graphics Color Values

Number	Color
0	Vision Video
1	Black
2	Dark gray
3	Blue
4	Blue gray
5	Light blue
6	Green
7	Dark green
8	Yellow
9	Orange
10	Red
11	Maroon
12	Pink
13	Medium gray
14	Light gray
15	White

Examples

For the window open on graphics logical unit 20, set the foreground color to black (1) and the background color to white (15) for subsequent graphics instructions:

```
GCOLOR (20) 1, 15
```

For the window open on graphics logical unit 20, set the foreground color to green (6) for subsequent graphics instructions (and do not change the background color):

```
GCOLOR (20) 6
```

Related Keywords

FOPEN (program instruction)

GARC (program instruction)

GCHAIN (program instruction)

GFLOOD (program instruction)

GLINE (program instruction)

GLINES (program instruction)

GPANEL (program instruction)

GPOINT (program instruction)

GRECTANGLE (program instruction)

GSCAN (program instruction)

GTYPE (program instruction)

Syntax

```
GCOPY (lun) x, y = src_x, src_y, dx, dy
```

Function

Copy one region of a window to another region in the same window.

Usage Considerations

This instruction is available only with an graphics-based system.

Parameters

lun	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
x, y	Real values, variables, or expressions (interpreted as integers) specifying the coordinates (in pixels) of the top left corner of the destination region.
src_x	Real values, variables, or expressions (interpreted as integers) specifying **src_y**the coordinates (in pixels) of the top left corner of the source region from which bitmap information is to be obtained.
dx, dy	Real values, variables, or expressions (interpreted as integers) specifying the width and height, respectively, of the region to copy. Both of these values must be greater than zero.

Details

The specified region of the window is copied directly to the destination region with the current logical operation applied (see GLOGICAL).

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

Example

The following instruction copies, in the window open on graphics logical unit 20, a 5-by-7 region from the rectangle with its top left corner at (10,15) to the rectangle with its top left corner at (0,2):

```
GCOPY(20) 0, 2 = 10, 15, 5, 7
```

Related Keywords

FOPEN (program instruction)

GCLIP (program instruction)

GLOGICAL (program instruction)

Syntax

```
GETC (lun, mode)
```

Function

Return the next character (byte) from a device or input record on the specified logical unit.

Usage Considerations

The logical unit must be attached by the program for normal, variable-length record input/output.

Parameters

lun Real value, variable, or expression (interpreted as an integer) that identifies the device to be accessed. (See the ATTACH instruction for a description of the unit numbers.)

mode Real value, variable, or expression (interpreted as an integer) that specifies the mode of the read operation. Currently, the mode is used only for the terminal and serial I/O logical units. The value is interpreted as a sequence of bit flags as detailed below. (All bits are assumed to be clear if no mode value is specified.)

Bit 1 (LSB) Disable waiting for input (mask value = 1)

If this bit is clear, program execution will be suspended until the next byte is received. If the bit is set and no bytes are available, the function will immediately return the error code for *No data received* (–526).

NOTE: A –526 error may be returned by the first no-wait GETC even if there are bytes queued.

Bit 2 Disable echo (mask value = 2)

If this bit is clear, input from the terminal is echoed back to the source. If the bit is set, characters are not echoed back to the source. (This bit is ignored for the serial lines.)

Details

The next byte from the device is returned. When reading from a record-oriented device such as the system terminal or a disk file, the carriage-return and line-feed characters at the end of records are also returned. When the end of a disk file is reached, a Ctrl+Z character (26 decimal) is returned.

When reading from the terminal, GETC will return the next character entered at the keyboard. All control characters will be read, except Ctrl+S, Ctrl+Q, Ctrl+O, and Ctrl+W, which will have their normal terminal control functions.

When reading from the serial line, GETC will return the next data byte immediately, unmodified. (Note that if the serial line is configured to recognize Ctrl+S and Ctrl+Q automatically as control characters, then those characters will not be returned by the GETC function.)

Normally, the byte value returned is in the range 0 to 255 (decimal). If an input error occurs, a negative error code number is returned. The meanings of the error codes are listed in **Appendix B**.

Example

The following program segment reads characters from a disk file until a comma (,) character, a control character, or an I/O error is encountered. The characters are appended to the string variable $field. (The disk file must have already been opened for accessing variable-length records.)

```
$field = ""
c = GETC(5)

WHILE (c > ^H1F) AND (c <> ',) DO
   $field = $field+$CHR(c)
   c = GETC(5)
END

IF c < 0 THEN
   TYPE $ERROR(c)
   HALT
END
```

Related Keywords

ATTACH (program instruction)

READ (program instruction)

Syntax

```
GET.EVENT (task)
```

Function

Return events that are set for the specified task.

Usage Considerations

Do not confuse GET.EVENT with the GETEVENT program instruction, which returns information from a graphics window or the terminal.

Parameter

task Optional real value, variable, or expression (interpreted as an integer) that specifies the task for which events are to be returned. The valid range is –1 to 6 or –1 to 27, inclusive.[1] If the parameter is omitted, or has the value –1, the current task is referenced.

Details

The events are returned in a value that should be interpreted as a sequence of bit flags, as detailed below.

Bit 1 (LSB) I/O Completion (mask value = 1)

This bit being set indicates that a system input/output operation has completed.

See the descriptions of SET.EVENT and WAIT.EVENT for more details.

Related Keywords

CLEAR.EVENT (program instruction)

SET.EVENT (program instruction)

WAIT.EVENT (program instruction)

[1] The basic system allows 7 tasks (0 - 6). The V+ Extensions option allows 28 tasks (0 - 27).

Syntax

GETEVENT (lun, mode**) events[**index**]**

Function

Return information describing input from a graphics window or input from the terminal.

Usage Considerations

A graphics window must be open for **read** access on the specified logical unit. If keyboard input is being read, the terminal must be attached.

> **NOTE:** Do not confuse GETEVENT with the GET.EVENT real-valued function, which returns events for a specified task.

Parameters

lun — Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)

mode — Optional real value, variable, or expression (interpreted as a bit field) specifying how the instruction is to operate. Zero is assumed if the parameter is omitted. The bit interpretations are given below.

Bit 1 (LSB)Wait (0) vs. No wait (1) (mask value = 1)

If this bit is clear, program execution will be suspended until the next event is received. If the bit is set and no events are available, the array will contain the error code for *No data received* (–526) (see note below).

> **NOTE:** A –526 error may sometimes be returned by a no-wait GETEVENT instruction even if there are already events queued. Subsequent GETEVENT instructions will return these events.

events[] — Real array that receives the event code number and data if any event is available. The format of the event information is described below.

index — Optional real value, variable, or expression (interpreted as an integer) specifying the **first** array element to receive data. Element zero is accessed first if no index is specified.

Details

The GETEVENT instruction operates on the window that is currently open for the specified logical unit. If the window is open for write-only access, an error will be reported and no event information will be available.

All input from a window—including movement of the pointing device, button presses, keyboard input, and window status changes—can be received in the form of events. Recognition of the various events is enabled for a window with the /EVENT attribute keyword for the FOPEN and FSET instructions (see below). Whenever an enabled event occurs for a window, the event is placed in an event queue for that window.

The GETEVENT instruction allows an application program to read events from the queues and process them. The information returned in the **`events[]`** array indicates the success or failure of the instruction and, if successful, which events occurred. The format of the **`events[]`** array is as follows:

Array element	Description of contents
index+0	Error code or event code If this value is less than zero, the instruction has failed and the value is a standard V+ error number. In this case, no other elements are significant. If this value is zero or positive, the value is a code that indicates which event has occurred. (See below for descriptions of the event codes.)
index+1	Event argument #1 (if any, see below)
...	
index+N	Event argument #N (if any, see below)

After a GETEVENT instruction is executed, the real-valued function IOSTAT(logical_unit, 2) returns the number of values placed in the **`events[]`** array, including the event code and any arguments.

The FCMND program instruction (with command code #102) can be used to discard all the events waiting to be processed.

If keyboard events are enabled for a window, all keypresses (except Crtl+W, Crtl+S, and Ctrl+Q) are returned whenever the window is selected.

GETEVENT can also be used to intercept keyboard input to the monitor window simply by ATTACHing the terminal and specifying the terminal LUN in the GETEVENT instruction (a window does not have to be open and events do not have to be enabled).

Event Codes

Each event is identified by an integer called the event code. This code tells which event has occurred. Events may have a number of associated arguments, and the application program must interpret them. Table 2-13. shows, for each event, the event code and the arguments. (Each argument listed below is a 16-bit integer value.)

Table 2-13. Graphics Events Codes

Code	Description	Arguments (see notes below)	
0	Keypress	key = ASCII value of key pressed	
1	Button down	button_mask, x, y	(See note 1)
2	Button up	button_mask, x, y	(See note 1)
3	Double click	button_mask, x, y	(See notes 1 & 2)
4	Pointer moved	button_mask, x, y	(See note 1)
5	Window select		
6	Window deselect		
7	Slide bar button down	ID, position, max_pos	(See note 3)
8	Slide bar pointer move	ID, position, max_pos	(See note 3)
9	Slide bar button up	ID, position, max_pos	(See note 3)
10	Size button down	dummy (zero), dx, dy	(See note 4)
11	Size pointer move	dummy (zero), dx, dy	(See note 4)
12	Size button up	dummy (zero), dx, dy	(See note 4)
13	Not used		
14	Menu selected	menu, item	(See note 5)
15	Pointer enter window		
16	Pointer exit window		

Table 2-13. Graphics Events Codes (Continued)

Code	Description	Arguments (see notes below)	
17	Close window		
18	Open window		
20	Connected to a PC		
21	Disconnected from a PC		

The following notes apply to **Table 2-13**:

Note 1: The button bit mask consists of a bit for each button on the pointing device. The mask values are 1 for button 1, 2 for button 2, and 4 for button 3. On three-button pointing devices, button 1 is the left button, 2 is the middle button, and 3 is the right button. On two-button devices (such as the Adept integrated keyboard), the left button is #2 and the right button is #3. For button down, button up, and double-click events, only a single bit is set in the mask, corresponding to the button that generated the event. For pointer-moved events, the bit mask reflects the current state of all three pointer device buttons. A set bit indicates that the corresponding button is pressed.

Note 2: A double-click event occurs in addition to all of the four other button events that created it. The complete sequence is: button down, button up, button down, double click, button up.

Note 3: These events are returned by the slide bars (see the GSLIDE instruction) contained in the window, as well as the horizontal and vertical scroll bars belonging to the window.

For user-defined slide bars, the ID number is supplied by the user in the GSLIDE instruction that creates the slide bar. The position is in the range 0 to max_pos (which is also supplied by the user in the GSLIDE instruction).

The ID number for the window vertical scroll bar is –1. If vertical scroll is **not** being grabbed (see the GRAB_V_SCROLL keyword below), the position is the actual vertical scroll in pixels. max_pos is the maximum possible vertical scroll in pixels. If vertical scroll **is** being grabbed, max_pos may be previously set by the user (see /V_RANGE under the FSET instruction). position may also be set by the user, although it will be modified by pointer device activity (see the attributes /V_HANDLE and /V_ARROWINC for the FSET instruction).

The ID number for the window horizontal scroll bar is –2. If horizontal scroll is **not** being grabbed (see the GRAB_H_SCROLL keyword below), the position is the actual horizontal scroll in pixels. max_pos is the maximum possible horizontal scroll in pixels. If horizontal scroll **is** being grabbed, max_pos may be previously set by the user (see /H_RANGE under the FSET instruction). position may also be set by the user, although it will be modified by pointer device activity (see the attributes /H_HANDLE and /H_ARROWINC for the FSET instruction).

Note 4: All the sizing events return the new window width (dx) and height (dy). At the time either or both of the NTFY_SIZE and GRAB_SIZE events are **enabled** (see /EVENT keywords below), a size button down event followed by a size button up event are returned with the current window size.

Note 5: The arguments for the menu-selected event should be interpreted as follows:

Menu	Description	Item
0	Menu-bar item selected	Item number of menu-bar item selected. (The program should issue an FSET /PULLDOWN ... in response.)
>0	Pull-down item selected	Row number of pull-down item selected, counting from 1.

Enabling Event Recognition

In order to improve system efficiency and minimize memory requirements, only events that are actually needed should be enabled. Events that are not enabled do not appear in any queue and are simply ignored.

Event recognition is controlled with the /EVENT attribute keyword specified in an FOPEN or FSET instruction. The argument for the /EVENT keyword is a list of event keywords, each of which identifies an event to be enabled or disabled. In general, each event keyword can be prefixed with NO to disable the event. If conflicting event keywords are encountered, the later keyword takes precedence.

The /EVENT keywords are shown below along with the names of the events they control. When a window is created, no events are enabled. Thus, the desired events must all be specified with the /EVENT keyword in an FOPEN or FSET instruction. Each of the keywords may be abbreviated as long as the abbreviation cannot be confused with another keyword's.

NOTE: Adept recommends that keywords **not** be abbreviated, to ensure that programs will be compatible with future releases of V+.

Keyword	Description & effect when enabled	Events enabled
BUTTON	Pointer button change (see Note 1 below)	Button down Button up Double click
GRAB_H_SCROLL	Horizontal scroll events go to program and do not change window (see Note 2 below)	Scroll horizontal
GRAB_OPEN	Window open and close events go to program and do not change the window	Open window Close window
GRAB_SIZE	Size change events go to program and do not change window (see Note 3 below)	Size change
GRAB_V_SCROLL	Vertical scroll events go to program and do not change window (see Note 2 below)	Scroll vertical
KEYPRESS	Keyboard input processed if the window is selected	Keypress
MENU	Pull-down menu selections	Menu selected
MOVE_ANY	Pointer-move event any time (not implemented)	Pointer move
MOVE_B1	Pointer move while button 1 down (see Note 1 below)	Pointer move
MOVE_B2	Pointer move while button 2 down (see Note 1 below)	Pointer move
MOVE_B3	Pointer move while button 3 down (see Note 1 below)	Pointer move
NONE	Disable all events (may not be preceded with NO)	None
NTFY_H_SCROLL	Horizontal scroll—events go to program and also change window	Scroll horizontal

Keyword	Description & effect when enabled	Events enabled
NTFY_OPEN	Window open and close—events go to program and also change window	Open window Close window
NTFY_SIZE	Size change—events go to program and also change window (see Note 3 below)	Size change
NTFY_V_SCROLL	Vertical scroll—events go to program and also change window	Scroll vertical
OBJECT	Event occurs when clicking and dragging within a slide bar (see Notes 1 and 4 below)	Slide bar
POINTER_CHANGE	Event occurs when pointer crosses window boundaries (not implemented)	Pointer enter window Pointer exit window
SELECT_WINDOW	Event occurs when select status of the window changes	Window select Window deselect

The following notes apply to the information above:

Note 1: If the OBJECT keyword is specified together with the BUTTON and MOVE_B2 keywords, the slide bar events may be returned as well as pointer-2 move and button-2 events. All events caused by button 2 that do not cause a slide bar event will be returned as pointer-move or button-2 events. On the other hand, if the cursor is within an active slide bar, only slide bar events will be returned.

Note 2: (The following paragraphs apply equally to the horizontal and vertical scroll bars. For the horizontal bar, substitute the keywords GRAB_H_SCROLL, /H_RANGE, /H_HANDLE, and /H_ARROWINC for GRAB_V_SCROLL, /V_RANGE, /V_HANDLE, and /V_ARROWINC, respectively.)

The vertical scroll bar operates in essentially two modes. In the first mode (in which the GRAB_V_SCROLL event has **not** been enabled), the vertical scroll bar is under the control of the window manager. In this mode, the maximum value of the

position returned by the vertical scroll bar is equal to the total possible window scroll (in pixels).

In the second mode, the vertical scroll bar is under the control of the application program. This occurs when the GRAB_V_SCROLL event has been enabled. In this mode, the maximum value of the position returned by the scroll bar can be set by using the FSET instruction with the /V_RANGE attribute. Additionally, the vertical scroll handle position and the position increment due to the arrow buttons can be set (see /V_HANDLE and V_ARROWINC attributes for the FSET instruction).

At the time the GRAB_V_SCROLL event is disabled, the vertical scroll bar will return to the control of the window manager. The maximum value of the position returned by the vertical scroll bar will again be equal to the total possible window scroll in pixels. The information set by /V_RANGE, /V_HANDLE, and /V_ARROWINC will be lost.

Note 3: At the time a GRAB_SIZE or NTFY_SIZE event is **enabled**, a size button down event and a size button up event are sent to the window. In this manner, a user program may obtain the size of a window as soon as either of these events is enabled.

Note 4: The slide-bar events can be caused only by button 2. Clicking and/or dragging on a slide bar with any other button will cause only pointer-move or button events (if they are enabled).

Example

Get the next event from the window currently open for read on graphics logical unit 20. Stores the event code and arguments in array elements events[0], events[1], etc.

```
GETEVENT (20) events[]
```

Related Keywords

FCMND (program instruction)

FOPEN (program instruction)

FSET (program instruction)

Syntax

```
GFLOOD (logical_unit) x, y
```

Function

Flood a region in a graphics window with color.

Usage Considerations

This instruction is available only with an graphics-based system.

This instruction is fairly slow and should be avoided in time-critical applications.

Parameters

`lun`	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
`x, y`	Real values, variables, or expressions (interpreted as integers) specifying the (pixel) coordinates of the seed point from which the flooding is to proceed.

Details

This instruction floods any region enclosing the given point (x,y) with the current graphics foreground color (see GCOLOR). The output from the GFLOOD instruction is not affected by the current texture or logical operation.

The flooding starts with the assumption that the color of the specified seed pixel at (x,y) is the old color that is to be replaced with the new color specified. As GFLOOD floods the area around (x,y) with the new color, it looks for region boundaries to stop the flood. Any pixel different from the initial old color is treated as a region boundary.

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

Example

In the window currently open on graphics logical unit 20, change the pixel at coordinates (100,120) to the current foreground color, and propagate that change to all surrounding pixels that have the original color of pixel (100,120).

```
GFLOOD (20) 100, 120
```

Related Keywords

FOPEN (program instruction)

GCOLOR (program instruction)

Syntax

```
GGETLINE (logical_unit) $data[index], num.pix = x, y, nx
```

Function

Return pixel information from a single pixel row in a graphics window.

Usage Considerations

Older versions of V+ provided the instruction GGET.LINE, which has a similar function. GGET.LINE is supported in V+ version 11.0, but will become obsolete in a future version. New code should use the syntax presented here. Existing code should be updated to use the new instruction name.

Parameters

lun	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
$data[]	String array that receives the returned pixel information. The format of the pixel information is described below.
index	Optional real value, variable, or expression (interpreted as an integer) specifying the **first** array element to receive data. Element zero is accessed first if no index is specified.
num.pix	Returns the number of pixels in the **$data[]** array. This will be less than the number of pixels requested if the edge of the window is encountered before **nx** pixels can be returned.
x, y	Real values, variables, or expressions (interpreted as integers) specifying the (pixel) coordinates of the starting point of the row of pixels to return.
nx	Real value, variable, or expression (interpreted as an integer) specifying the number of pixels to return.

Details

The GGETLINE instruction operates on the window that is currently open for the specified logical unit.

This instruction allows an application program to retrieve pixel information from a graphics window. The information returned in the string array describes **nx** pixels in row **y** of the graphics window, starting at column **x**. The format of the string array is as follows:

- For windows with 4 bits per pixel (/COLORS 16):

Table 2-14. String Arrays with 4 Bits per Pixel

	Byte 1 of $data[index]								Byte 2 of $data[index]								etc.
Bit #:	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	etc.
Pixel #:	x				x+1				x+2				x+3				etc.

- For windows with 1 bit per pixel (/COLORS 2):

Table 2-15. String Arrays with 1 Bit per Pixel

	Byte 1 of $data[index]								Byte 2 of $data[index]								etc.
Bit:	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	etc.
Pixel #:	x	x+1	x+2	x+3	x+4	x+5	x+6	x+7	x+8	x+9	x+10	x+11	x+12	x+13	x+14	x+15	etc.

If more than 128 bytes of data are returned, the 129th data byte is returned as the first byte of $data[index+1], the 257th data byte is returned as the first byte of $data[index+2], and so forth.

All the elements of **`$data[]`** filled with data will be 128 bytes long, except for the last element filled (which will be from zero to 127 bytes long). This fact can be used to determine the number of bytes actually containing pixel data (see the example below).

Example

The following instruction will access the window opened on logical unit number 20. Pixel data from the window will be placed in array elements $data[0] and $data[1], starting from the pixel at coordinates (100,50). Data for 250 pixels will be returned (unless the right margin of the window is encountered first).

```
GGETLINE (20) $data[] = 100, 50, 250
```

The following program segment shows how to compute the actual number of bytes of pixel data returned in the array $data[].

```
index = 0     ;First array index
bytes = 0     ;Number of data bytes

DO
   length = LEN($data[index]) ;Length of this element
   bytes = bytes+length       ;Total the bytes
   index = index+1            ;On to the next element
UNTIL length < 128            ;Partial string signals end
```

Related Keyword

FOPEN (program instruction)

Syntax

```
GGET.LINE (logical_unit) $data[index], num.pix = x, y, nx
```

Function

Return pixel information from a single pixel row in a graphics window.

Usage Considerations

GGET.LINE is supported in V+ version 12.1, but will become obsolete in a future version. New code should use the syntax presented here. Existing code should be updated to use GGETLINE.

Parameters

See the GGETLINE program instruction

Details

See the GGETLINE program instruction

Example

See the GGETLINE program instruction

Related Keyword

FOPEN (program instruction)

GGETLINE (program instruction)

Syntax

```
GICON (lun, mode) x, y, $name, index
```

Function

Draw a predefined graphic symbol (icon) in a graphics window.

Usage Considerations

This instruction is available only with an graphics-based system.

The specified icon must be either a predefined system icon or a defined user icon.

Parameters

lun	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
mode	Optional real value, variable, or expression (interpreted as an integer) specifying the icon drawing mode. The mode values are as follows:
0	= Draw a static icon using the actual icon colors, overwriting the display. (This is the default.)
1	= Draw a static icon using the actual icon colors, overwriting the display, but draw only every other pixel. (The resulting icon image looks like it has vertical, transparent stripes.)
2	= Draw a static icon in complement mode—the icon image is exclusive ORed with the current window contents. (Note that drawing an icon a second time in this mode will restore the previous background.)
3	= Erase the current movable icon, if any. The **x**, **y**, **$name**, and index parameters are ignored in this mode.
4	= After erasing any previous movable icon, draw the icon in complement mode. The new icon becomes the current movable icon.

x, y	Real values, variables, or expressions (interpreted as integers) specifying the (pixel) coordinates of where the icon is to be drawn. The icon reference point is placed at these coordinates. The location of the reference point in the icon depends upon the particular icon, but it is normally the top left corner or the center of the icon.
$name	String value, variable, or expression specifying the name of the icon to be drawn. (Icon names follow the same convention as normal V+ variable names.)
index	Optional real value, variable, or expression (interpreted as an integer) specifying which element is to be drawn from an array of icons. The value may range from 0 to 255. Element zero is drawn if no index is specified, or if the specified index is larger than the last array element defined.

Details

This instruction draws a system-defined or user-defined graphic symbol in the current window associated with the indicated logical unit number. The symbol, called an icon, consists of a rectangular region of pixels with arbitrary colors. The symbol is referenced by name, and may be either a standard system icon (see the list below) or a user-defined icon.

Icon name	Max index	Description
CHECK_BOX	3	Check box for on/off selection
CURSOR	0	Pointing-device cursor
RADIO_BUTTON	3	Push button for on/off selection
SYSTEM_ADEPT		Adept icon in screen upper left
SYSTEM_CLOSE	2	Close icon in window upper left
SYSTEM_CURSOR	6	Cursors
SYSTEM_DOWN	1	Down arrow
SYSTEM_H_HANDLE	1	Handle for horizontal scroll bar
SYSTEM_LEFT	1	Left arrow
SYSTEM_MINMAX	1	Min/max icon above vert. scroll bar
SYSTEM_NULL		Null icon

Icon name	Max index	Description
SYSTEM_RIGHT	1	Right arrow
SYSTEM_SCROLL	2	Scroll bar display/suppress icon
SYSTEM_SIZE	1	Size icon in window lower right
SYSTEM_UP	1	Up arrow
SYSTEM_V_HANDLE	1	Handle for vertical scroll bar

Custom icons can be developed and stored in disk files with the Adept Icon Editing Utility program.[1] At runtime, your application program can call the program load.icon to read the data files and define the icons in graphics memory. Once they are loaded into graphics memory, the icons can be displayed with the GICON instruction.

Custom icons also can be created directly by using the FOPEN instruction to open an icon (see the FOPEN instruction for a description of the /ICON attribute) and then using the WRITE instruction to write image data into graphics memory. After an icon has been opened for definition, the WRITE instruction must be used repeatedly to define each row of the icon bitmap. The string supplied to the WRITE instruction must contain the following items in the order listed:

1. A 16-bit integer indicating the index of the icon being defined [for example, defined by $INTB(index)]
2. A 16-bit integer indicating the row of the icon being defined [for example, defined by $INTB(row)]
3. Enough four-bit pixel values to define a row of the icon—data in excess of that needed to define a row will be ignored. For example, the value for a **pair** of pixels could be defined by an expression like:

   ```
   $CHR(^H10*first_pixel+second_pixel)
   ```

4. The /S format control specifier—to prevent carriage-return and line-feed characters from being appended to the string

A one-bit-per-pixel version of the icon also will be defined automatically. All zero pixels in the four-bit-per-pixel icon will appear as 0 pixels in the one-bit-per-pixel icon, and all nonzero pixels will appear as 1 pixels.

User-defined icons can be deleted from the graphics memory by using the /ICON qualifier in an FDELETE instruction. See FDELETE for details.

[1] The Icon Editing Utility is available in AIM software packages and on the Adept Applications Disk. Contact Adept Applications Dept. at (800) 232-3378.

Icons may be stored and referenced as arrays. Then the optional index parameter indicates which of the array elements is to be displayed. All the icons in an array have the same size. For arrays of icons that are to be selected via a mouse button press, the Adept convention is to use an even numbered array index (for example, 0) for the icon displayed when the button is not pressed, and the next index (for example, 1) for the corresponding button-down icon.

If there is no icon defined with the specified name and index, the system icon SYSTEM_NULL is displayed and no error is returned. This icon appears (when mode is 0) as a large red question mark.

Icons are not affected by the current logical operation, texture, or pattern. Depending on the mode parameter, icons may be either static or movable. Static icons are drawn in normal mode, using the actual icon colors, or in complement mode (see below). Movable icons are always drawn in complement mode. V+ keeps track of the movable icon in each window, and erases it (restoring the background) in modes 3 and 4. Only one movable icon is available in each window.

When an icon is drawn in complement mode, the bitmap for the icon is exclusive ORed with the bitmap for the window (using the current GLOGICAL planes mask), so the colors displayed will normally be different from the actual icon colors.

Pixel color zero is assumed to be transparent and allows the background to show. Thus, live video cannot be displayed within an icon unless the background is already live video.

The GICON instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the icon will be displayed when the buffer fills or an FEMPTY instruction is executed.

Example

The following instructions draw system-defined icons in the window currently open on logical unit number 20. The first instruction draws the system-defined up-arrow icon (named SYSTEM_UP), with its upper left-hand corner at location (10,15). The second instruction draws the button-down version of the same icon, in complement mode, to the right of the first icon:

```
GICON (20) 10, 15, "SYSTEM_UP"

GICON (20, 2) 50, 15, "SYSTEM_UP", 1
```

Related Keywords

FDELETE (program instruction)

FOPEN (program instruction)

GCLIP (program instruction)

Syntax

```
GLINE (lun) x0, y0, xn, yn
```

Function

Draw a single line segment in a graphics window.

Usage Considerations

This instruction is available only with an graphics-based system.

Parameters

lun	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
x0, y0	Real values, variables, or expressions (interpreted as integers) specifying the (pixel) coordinates of the starting point of the line.
xn, yn	Real values, variables, or expressions (interpreted as integers) specifying the (pixel) coordinates of the ending point of the line.

Details

The line is drawn with the current graphics foreground color (see GCOLOR), texture (see GTEXTURE), and logical operation (see GLOGICAL).

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

Example

In the window currently open on graphics logical unit 20, draw a line from coordinates (100,110) to (400,450), using the current foreground color:

```
GLINE (20) 100, 110, 400, 450
```

Related Keywords

FOPEN	(program instruction)
GARC	(program instruction)
GCLIP	(program instruction)
GCOLOR	(program instruction)
GLINES	(program instruction)
GLOGICAL	(program instruction)
GTEXTURE	(program instruction)
GTRANS	(program instruction)

Syntax

```
GLINES (logical_unit, mode) points, coord[offset,index]
```

Function

Draw multiple line segments in a graphics window.

Usage Considerations

This instruction is available only with an graphics-based system.

Parameters

lun Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)

mode Optional real value, variable, or expression (interpreted as a bit field) specifying how the instruction is to operate. Zero is assumed if the parameter is omitted. The bit interpretations are given below.

Bit 1 (LSB)Empty (0) vs. Filled (1)Mask value = 1

If this bit is set **and** bit #2 is set, the polygon is filled with the foreground color. (Not implemented)

Bit 2 Open (0) vs. Closed (2)Mask value = 2

If this bit is set, a line is drawn from the last point to the first to form a closed polygon.

points Real value, variable, or expression (interpreted as an integer) specifying the number of points to connect with lines.

coord[,] Real array containing pairs of position coordinates (in pixels) for the points. The array indexes are described below. The array elements are interpreted as follows:

Array element	Interpretation of value
coord[offset+0,index+0]	X coordinate for first point
coord[offset+1,index+0]	Y coordinate for first point
coord[offset+0,index+1]	X coordinate for second point
coord[offset+1,index+1]	Y coordinate for second point

`offset` Optional real value, variable, or expression (interpreted as an integer) specifying the left-hand array index value that accesses the X components of the coordinates for the points to be connected. Zero is assumed if this parameter is omitted—in which case `index` also must be omitted.

The X value is accessed using a left-hand index equal to `offset`. The Y value is accessed using a left-hand index equal to (offset+1).

`index` Optional real value, variable, or expression (interpreted as an integer) specifying the array elements to access for the **first** point in the series. Zero is assumed if this parameter is omitted. The maximum number of elements allowed is 127.

Details

This instruction draws a series of lines between the specified points. The mode parameter provides the option of having the last point automatically connected to the first point, forming a closed polygon. The lines are drawn with the current graphics foreground color (see GCOLOR), texture (see GTEXTURE), and logical operation (see GLOGICAL).

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

Examples

The following instruction draws lines connecting a series of 10 points in the window currently open on graphics logical unit 20, using the foreground color. The endpoints of the first line have coordinates (lines[1,1],lines[2,1]) and (lines[1,2], lines[2,2]). The endpoints of the last line are (lines[1,9],lines[2,9]) and (lines[1,10],lines[2,10]):

```
GLINES (20) 10, lines[1,1]
```

The following instruction draws a closed polygon with 5 sides in the window currently open on graphics logical unit 20, using the foreground color. The endpoints of the first line have coordinates (lines[0,0], lines[1,0]) and (lines[0,1],lines[1,1]). The endpoints of the last line are (lines[0,4],lines[1,4]) and (lines[0,0], lines[1,0]):

```
GLINES (20, 2) 5, lines[,]
```

Related Keywords

FOPEN (program instruction)

GCLIP (program instruction)

GCOLOR (program instruction)

GLINE (program instruction)

GLOGICAL (program instruction)

GRECTANGLE (program instruction)

GSCAN (program instruction)

GTEXTURE (program instruction)

GTRANS (program instruction)

Syntax

```
GLOBAL type variable, ..., variable
```

Function

Declare a variable to be global and specify the type of the variable.

Parameters

`type`	Optional parameter specifying the type of a variable. The acceptable types are:
LOC	Location variable (transformation, precision point, belt)
REAL	Single-precision real variable
DOUBLE	Double-precision real variable
	See the Details section for the default type.
`variable`	Variable name (belt, precision point, real-value, string, and transformation). Each variable can be a simple variable or an array. If the `type` parameter is specified, all the variables must match the specified type. Array variables must have their indexes specified explicitly, indicating the highest valid index for the array.

Details

Variables that are not declared to be AUTO or LOCAL are GLOBAL by default. Undeclared scalar variables default to single precision.

Thus, single-precision and location global variables do not need to be declared. However, the only way to guarantee a double-precision global variable is with the GLOBAL instruction.

Global variables can be seen by any program that does not declare a LOCAL or AUTO variable of the same name. Thus, if **program_a** declares var1 to be a GLOBAL variable and **program_b** declares var1 to be AUTO, **program_b** will not be able to use or alter GLOBAL var1. A new copy of variable var1 that is specific to **program_b** will be created each time **program_b** executes.

GLOBAL DOUBLEs must be declared in each program in which they are used.

Examples

```
GLOBAL $str_1, $str_2, str_3 ;create 2 string and 1 untyped
                             ;variable
GLOBAL LOC #ppoint_1         ;create 1 global precision
                             ;point variable
GLOBAL DOUBLE var_1, var_2   ;create 2 double prec. reals
```

Related Keywords

AUTO (program instruction)

LOCAL (program instruction)

Syntax

```
GLOGICAL (logical_unit) code, planes
```

Function

Set the logical operation to be performed between new graphics output and graphics data already displayed, and select which bit planes are affected by graphics instructions.

Usage Considerations

This instruction is available only with an graphics-based system.

The logical operation and the bit-plane settings are reset for the specified logical unit when a window is opened on the logical unit with the FOPEN instruction.

Parameters

lun Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)

code Optional real value, variable, or expression (interpreted as an integer) specifying the function code of the logical operation to be performed on graphics data as it is output to the display. The specified operation, as described in the table below, is performed with the new output (source) and any previous output already in the window (dest). The value 3 is used if this parameter is omitted. (Most of these logical operations are used only in unusual circumstances.)

code	Logical operation selected
0	0 [set dest bits to 0]
1	source AND dest
2	source AND (NOT dest)
3	source
4	(NOT source) AND dest
5	dest [do not change dest]
6	source XOR dest

code	Logical operation selected
7	source OR dest
8	(NOT source) AND (NOT dest)
9	(NOT source) XOR dest
10	NOT dest
11	source OR (NOT dest)
12	NOT source
13	(NOT source) OR dest
14	(NOT source) OR (NOT dest)
15	1 [set dest bits to 1]

`planes` Optional real value, variable, or expression (interpreted as a 1-bit or 4-bit field) specifying the bit planes that are to be accessed during subsequent graphics output. For each bit that is set, the corresponding bit plane is enabled for writing during subsequent commands. All bit planes will be enabled if this parameter is omitted.

Details

This instruction establishes the logical operation to be performed between the output requested by a graphics instruction and the existing graphics output already in the window. That is, the logical operation selected will be performed for all subsequent graphics output, until another GLOGICAL instruction is executed.

GLOGICAL also selects which bit planes are affected by graphics output. In a 16-color window, the color at each pixel is represented by 4 bits. The planes parameter restricts graphics operations to changing only certain bits within each pixel. The color of a pixel as seen on the screen then depends upon the combination of previous unchanged bit values and any new bit values. Normally, all bit planes should be written.

Output from all the following instructions is affected by the current GLOGICAL settings:

GARC	GCHAIN	GFLOOD	GLINE	GLINES
GPOINT	GRECTANGLE	GSCAN	GTYPE	

This instruction operates on the window that is currently open for the specified logical unit. The effect of this instruction will apply to all subsequent output to the window. (Note, however, that if the window is open in buffered mode, the effect will not be displayed until the buffer fills or an FEMPTY instruction is executed.)

Examples

For the window open on graphics logical unit 20, select the 3 low-order bits (mask 7) of each pixel to be written directly from the 3 low-order bits of the graphics source (function code 3) for subsequent graphics instructions.

```
GLOGICAL (20) 3, 7
```

For the window open on graphics logical unit 20, specify that for subsequent graphics output all bits of each pixel will be written as the exclusive OR of the existing pixel value with the new value from the graphics output (function code 6):

```
GLOGICAL (20) 6
```

For the window open on graphics logical unit 20, cancel any special logical operation or bit plane selections, so that subsequent graphics instructions will write all pixels normally:

```
GLOGICAL (20)
```

Related Keywords

FOPEN (program instruction)

GARC (program instruction)

GCHAIN (program instruction)

GCOPY (program instruction)

GLINE (program instruction)

GLINES (program instruction)

GLOGICAL (program instruction)

GPOINT (program instruction)

GRECTANGLE (program instruction)

GSCAN (program instruction)

GTYPE (program instruction)

Syntax

GOTO label

Function

Perform an unconditional branch to the program step identified by the given label.

Parameter

label	Label of the program step to which execution is to branch. Step labels are integer values that range in value from 0 to 65535.

Details

This instruction causes program execution to jump to the line that contains the specified step label. Note that a step **label** is different from a line number. Line numbers are the numbers automatically assigned by the V+ program editors to assist the editing process. Step labels must be explicitly entered on program lines where appropriate.

Modern, structured programming considers GOTO statements to be poor programming practice. Adept suggests you use one of the other control structures in place of GOTO statements.

Example

The following program segment asks the user to enter a number from 1 to 100. If the number input is not in that range, the GOTO 10 instruction at line number 27 causes execution to jump to step **label** 10 (at line number 23).

```
21 ; Get a number from the user
22
23 10 PROMPT "Enter a number from 1 to 100: ", number
24
25   IF (number < 1) OR (number > 100) THEN
26     TYPE /B, /C1, *Invalid response*, /C1
27     GOTO 10
28   END
```

Related Keywords

DO	(program instruction)
EXIT	(program instruction
FOR	(program instruction)
IF GOTO	(program instruction)
IF ... THEN	(program instruction)
NEXT	(program instruction)
WHILE	(program instruction)

Syntax

GPANEL (lun, mode**) x, y, dx, dy**

Function

Draw a rectangular panel with shadowed or grooved edges.

Usage Considerations

This instruction is available only with an graphics-based system.

Parameters

lun Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)

mode Optional real value, variable, or expression (interpreted as a bit field) specifying how the instruction is to operate. Zero is assumed if the parameter is omitted. The bit interpretations are given below.

Bit 1 (LSB) Empty (0) vs. Filled (1) (mask value = 1)

If this bit is set, the panel area is filled with the current foreground color.

Bit 2 Raised (0) vs. Sunken (1) (mask value = 2)

If this bit is set, the panel edges will be shaded to appear depressed below, rather than raised above, the surrounding area.

Bit 3 Panel (0) vs. Groove (1) (mask value = 4)

If this bit is set, a rectangular groove or ridge (depending on bit #2) two pixels wide will be drawn around the panel area instead of a shaded edge.

x, y Real values, variables, or expressions (interpreted as integers) specifying the coordinates (in pixels) of the interior pixel in the top left corner of the rectangular panel. That is, these parameters locate the top left-most pixel on the surface of the panel.

dx, dy Real values, variables, or expressions (interpreted as integers) specifying the width and height, respectively, of the panel surface or the area within the grooves. Both of these values must be greater than zero.

Details

This instruction draws a rectangular panel in the graphics window, similar to the output from the GRECTANGLE instruction. The panel is drawn (and optionally filled) with the current graphics foreground color (see GCOLOR). Unlike GRECTANGLE, however, the GPANEL instruction has the following features:

- The edges of the rectangle are shadowed to make it appear raised or depressed relative to the surrounding area.
- The edges of the rectangle can be made to appear as a groove or ridge enclosing the rectangle, rather than raising or depressing the surface of the rectangle.
- The current graphics logical operation (as set by GLOGICAL) is not performed.
- The current graphics texture settings (as set by GTEXTURE) are ignored.

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

Examples

In the window currently open on graphics logical unit 20, draw a standard raised panel at coordinates (10,15) with width 50 and height 20:

```
GPANEL (20) 10, 15, 50, 20
```

In the window currently open on graphics logical unit 20, draw a sunken panel, filled with the foreground color, at coordinates (10,50) with width 60 and height 30:

```
GPANEL (20,3) 10, 50, 60, 30
```

In the window currently open on graphics logical unit 20, draw a ridge around a rectangular area at coordinates (10,60) with width 40 and height 20:

```
GPANEL (20,6) 10, 60, 40, 20
```

Related Keywords

FOPEN	(program instruction)
GCLIP	(program instruction)
GCOLOR	(program instruction)
GRECTANGLE	(program instruction)
GTRANS	(program instruction)

Syntax

```
GPOINT (lun) x, y
```

Function

Draw a single point in a graphics window.

Usage Considerations

This instruction is available only with an graphics-based system.

Parameters

lun	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
x, y	Real values, variables, or expressions (interpreted as integers) specifying the coordinates (in pixels) of the point to be output.

Details

The point is drawn with the current graphics foreground color (see GCOLOR) and logical operation (see GLOGICAL).

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

Example

In the window currently open on graphics logical unit 20, draw a single point with the foreground color at coordinates (100,110):

```
GPOINT (20) 100, 110
```

Related Keywords

FOPEN	(program instruction)
GCHAIN	(program instruction)
GCLIP	(program instruction)
GCOLOR	(program instruction)
GLOGICAL	(program instruction)
GTRANS	(program instruction)

Syntax

GRECTANGLE (lun, mode**) x, y, dx, dy**

Function

Draw a rectangle in a graphics window.

Usage Considerations

This instruction is available only if the system controller includes a graphics system processor.

Parameters

lun	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
mode	Optional real value, variable, or expression (interpreted as a bit field) specifying how the instruction is to operate. Zero is assumed if the parameter is omitted. The bit interpretations are given below. Bit 1 (LSB)Empty (0) vs. Filled (1)(mask value = 1) If this bit is set, the rectangle is filled with the foreground color.
x, y	Real values, variables, or expressions (interpreted as integers) specifying the coordinates (in pixels) of the top left corner of the rectangle.
dx, dy	Real values, variables, or expressions (interpreted as integers) specifying the width and height, respectively, of the rectangle. Both of these values must be greater than zero.

Details

The rectangle is drawn and/or filled with the current graphics foreground color (see GCOLOR), texture (see GTEXTURE), and logical operation (see GLOGICAL).

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

Examples

In the window currently open on graphics logical unit 20, draw a solid rectangle in the foreground color at coordinates (10,20), with width 30 and height 40:

```
GRECTANGLE (20,1) 10, 20, 30, 40
```

In the window currently open on graphics logical unit 20, draw a rectangle outline in the foreground color at coordinates (75,80), with width 40 and height 50:

```
GRECTANGLE (20) 75, 80, 40, 50
```

Related Keywords

FOPEN (program instruction)

GCLIP (program instruction)

GCOLOR (program instruction)

GLOGICAL (program instruction)

GPANEL (program instruction)

GTEXTURE (program instruction)

GTRANS (program instruction)

Syntax

GSCAN (lun) lines, data[offset,index**]**

Function

Draw a number of horizontal lines in a graphics window to form a complex figure.

Usage Considerations

This instruction is available only with an graphics-based system.

Parameters

lun Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)

lines Real value, variable, or expression (interpreted as an integer) specifying the number of scan lines to draw.

data[,] Real array containing sets of data values (in pixels) for the lines. The array indexes are described below. The array elements are interpreted as follows:

Array element	Interpretation of value
data[offset+0,index+0]	X coord. for 1st start point
data[offset+1,index+0]	Y coord. for 1st start point
data[offset+2,index+0]	Length of first line
data[offset+0,index+1]	X coord. for 2nd start point
data[offset+1,index+1]	Y coord. for 2nd start point
data[offset+2,index+1]	Length of second line

`offset` Optional real value, variable, or expression (interpreted as an integer) specifying the left-hand array index value that accesses the X components of the coordinates for the start points. Zero is assumed if this parameter is omitted—in that case `index` must also be omitted.

The X value is accessed using a left-hand index equal to `offset`. The Y value is accessed using a left-hand index equal to

(offset+1). The length value is accessed using a left-hand index equal to (offset+2).

`index` Optional real value, variable, or expression (interpreted as an integer) specifying the array elements to access for the *first* line in the series. Zero is assumed if this parameter is omitted.

Details

This instruction draws a number of horizontal lines in the window to form a complex figure. The scan lines are drawn with the current graphics foreground color (see GCOLOR), texture (see GTEXTURE), and logical operation (see GLOGICAL).

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

Examples

In the window currently open on graphics logical unit 20, draw 25 horizontal lines in foreground color. The first line starts at (scan[0,0],scan[1,0]) and has length scan[2,0]. The last line starts at (scan[0,24],scan[1,24]) and has length scan[2,24]:

```
GSCAN (20) 25, scan[,]
```

In the window currently open on graphics logical unit 20, draw 15 horizontal lines in foreground color. The first line starts at (scan[2,3],scan[3,3]) and has length scan[4,3]. The last line starts at (scan[2,17],scan[3,17]) and has length scan[4,17]:

```
GSCAN (20) 15, scan[2,3]
```

Related Keywords

FOPEN (program instruction)

GCLIP (program instruction)

GCOLOR (program instruction)

GLOGICAL (program instruction)

GTEXTURE (program instruction)

Syntax

```
GSLIDE (lun, mode) id = x, y, length, max_pos, arrow_inc,
handle
```

Function

Draw a slide bar in preparation for receiving slide events.

Usage Considerations

This instruction is available only with a graphics-based system.

Parameters

lun Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)

mode Optional real value, variable, or expression (interpreted as a bit field) specifying how the instruction is to operate. Zero is assumed if the parameter is omitted. The bit interpretations are given below.

Bit 1 (LSB)Create/Update (0) vs. Delete (1)(mask value = 1)

The slide bar is deleted if this bit is set. Otherwise, the slide bar is created, or it is updated if it already exists.

Bit2 Horizontal (0) vs. Vertical (1)(mask value = 2)

If this bit is set, the slide bar will be created as a vertical slide bar. Otherwise, the slide bar will be horizontal. This bit is ignored if the slide bar already exists.

id Real value, variable, or expression (interpreted as an integer) specifying the ID number of the slide bar. This number is returned with slide event data to identify the slide bar to which the data applies. The value cannot be negative.

NOTE: All the following parameters (except for handle) must be specified if the slide bar is being created. If the slide bar already exists, these parameters can be omitted and (except for handle) will be ignored if they are specified.

`x, y`	Optional real values, variables, or expressions (interpreted as integers) specifying the (pixel) coordinates of the top left corner of the slide bar. Omitted values default to zero.
`length`	Real value, variable, or expression (interpreted as an integer) specifying the width of a horizontal slide bar or the height of a vertical slide bar. The value cannot be negative.
`max_pos`	Real value, variable, or expression (interpreted as an integer) specifying the value to be associated with the maximum handle position in the slide bar (which is the right end of a horizontal slide bar and the bottom end of a vertical slide bar). This parameter also determines how many positions the slide-bar handle can have in the slide bar (see below). If **`max_pos`** is zero, the slide bar will be drawn without a handle and the slide bar will be inactive. The value cannot be negative. The maximum value is 65,535.
`arrow_inc`	Real value, variable, or expression (interpreted as an integer) specifying the change in the position of the slide-bar handle that will occur for each mouse click on one of the arrow buttons (see below). The value cannot be negative.
`handle`	Optional real value, variable, or expression (interpreted as an integer) specifying the position of the handle (see below). The slide-bar handle position will be set even if the slide bar already exists. If `handle` is greater than **`max_pos`**, **`max_pos`** will be used. Zero will be used if this parameter is omitted or its value is negative.

Details

This instruction creates or adjusts a user-defined slide bar in a graphics window. The slide bar has the same appearance as the scroll bars V+ displays on the edges of windows.

If slide bar events are enabled (see the OBJECT event keyword described with the GETEVENT instruction), the slide bar will return events caused by clicking and/or dragging the mouse cursor on the slide bar. If slide bar events are disabled (or the **max_pos** parameter is specified to be zero), the slide bar will be inactive. That is, the slide-bar graphics will not reflect mouse cursor activity, and no events will be returned.

When the slide bar is active, the slide-bar handle can be moved to discrete positions in the slide bar, numbered from 0 to **max_pos**. If the user moves the handle to a position between two discrete positions, the handle will automatically jump to the nearest discrete position when the mouse button is released. The parameter **arrow_inc** establishes how many discrete positions the handle will move when the user clicks on one of the arrow buttons. The `handle` parameter sets the discrete position at which the handle is to be drawn.

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the slide bar will be displayed when the buffer fills or an FEMPTY instruction is executed.

Related Keywords

FOPEN (program instruction)

FSET (program instruction)

GCLIP (program instruction)

Syntax

```
GTEXTURE (lun) mode, pattern
```

Function

Set the opaque/transparent mode and the texture pattern for subsequent graphics output.

Usage Considerations

This instruction is available only with an graphics-based system.

The texture mode and pattern are reset for the specified logical unit when a window is opened on the logical unit with the FOPEN instruction.

Parameters

lun	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
mode	Optional real value, variable, or expression (interpreted as an integer) specifying that graphics output will be opaque (mode = 0) or transparent (mode = 1). The value 1 is assumed if this parameter is omitted.
pattern	Optional real value, variable, or expression (interpreted as a 16-bit integer) specifying the texture pattern. A solid pattern (corresponding to ^HFFFF) is assumed if this parameter is omitted.

Details

When characters or textured graphics are drawn, the contents of the window in the spaces behind the graphics can be left unchanged (transparent mode) or set to the current background color (opaque mode). For nontextured, noncharacter graphics, this mode has no effect. The interiors of filled geometric figures (drawn with mode bit 1 set) are always drawn solid, regardless of the texture.

The texture pattern allows dotted or dashed graphics to be drawn. The specified pattern is logically ANDed with the normal graphics output. This pattern is repeated every 4 pixels (in 16-color windows) or 32 pixels (in 2-color windows) along the geometric object drawn. The pixels that correspond to bits set in the pattern are drawn with the foreground color. The pixels that correspond to bits clear in the pattern are either set to the background color (in opaque mode) or are unchanged (in transparent mode).

The following instructions are affected by the GTEXTURE opaque/transparent mode setting and texture pattern, with exceptions as noted.

GARC (always in opaque mode)
GCHAIN (always in opaque mode)
GLINE
GLINES
GRECTANGLE
GSCAN
GTYPE (not affected by pattern)

This instruction operates on the window that is currently open for the specified logical unit. The effect of this instruction will apply to all subsequent output to the window. (Note, however, that if the window is open in buffered mode, the effect will not be displayed until the buffer fills or an FEMPTY instruction is executed.)

Examples

The following two instructions will draw a vertical dashed line, using the foreground color, on the current contents of the window currently open on graphics logical unit 20:

```
GTEXTURE (20) 1, ^HF0F0

GLINE (20) 50, 40, 50, 100
```

These two instructions draw a rectangle in the window currently open on graphics logical unit 21 (dotted lines are drawn, alternating between the foreground and background colors):

```
GTEXTURE (21) 0, ^H3333

GRECTANGLE(21) 30, 40, 150, 160
```

Related Keywords

FOPEN	(program instruction)
GARC	(program instruction)
GCHAIN	(program instruction)
GLINE	(program instruction)
GLINES	(program instruction)
GRECTANGLE	(program instruction)
GSCAN	(program instruction)
GTYPE	(program instruction)

Syntax

GTRANS (lun, mode**)** array[,]

Function

Scale, rotate, offset, and apply perspective correction to all subsequent graphics instructions.

Usage Considerations

The mode values 1 and 2 can be used only in systems equipped with the AdeptVision VXL option. With vision systems, this instruction allows you to specify graphics output to the vision window in real-world millimeters.

Parameters

lun	Integer specifying the graphics logical unit to apply the graphics transformation to.
mode	Integer specifying operational mode:
0	Use values in array[,] (default mode)
1	Use the camera calibration data in effect for the last camera that altered the vision system display frame store. Do not use the perspective distortion correction matrix.
2	Use the camera calibration data in effect for the last camera that altered the vision system display frame store. Use the perspective distortion correction matrix.
array[,]	When mode = 0, the values in array[,] are used to transform the position and orientation of subsequent graphics instructions.

Details

If `mode`and `array[,]` are omitted, the vision transformation is canceled. When `mode` = 0, the values in `array[,]` are used to transform the position and orientation components of subsequent graphics instructions. The affected instructions are listed in Table 2-16..

Table 2-16. Instructions Affected by GTRANS

Instruction and Parameters	Parameters Affected
GARC (lun, mode) xc, yc, radius, ang0, angn	The variables **xc** and **yc** are transformed. The value of **radius** is multiplied by the scaling. The values of **ang0** and **angn** are changed by the rotation angle.
GCHAIN (lun) x, y, n, points[n]	Only **x** and **y** are transformed.
GFLOOD (lun) x, y	Both **x** and **y** are transformed.
GICON (lun, mode) x, y, $name, index	x and y are transformed.
GLINE (lun) x0, y0, xn, yn	x0, y0, xn, yn are transformed.
GLINES (lun, mode) n, points[2,n]	Every array[0,n] and array[1,n] are transformed in pairs.
GPANEL (lun, mode) x, y, dx, dy	Only x and y are transformed.
GPOINT (lun) x, y	x and y are transformed.
GRECTANGLE (lun, mode) x, y, dx, dy	Only x and y are transformed.
GSCAN (lun) n, array[3,n]	Every array[0,n] and array[1,n] are transformed in pairs.
GSLIDE (lun, mode) id = x, y, length, num_pos, arrow_inc, handle	Only x and y are transformed.
GTYPE (lun, mode) x, y, $text, font_number, path, rotation	Only x and y are transformed.

Table 2-16. Instructions Affected by GTRANS (Continued)

Instruction and Parameters	Parameters Affected
GCLEAR GCLIP GCOLOR GCOPY GLOGICAL GTEXTURE	Not affected.

In modes 1 and 2, the camera calibration for the most recently used camera supplies the transformation matrix. See the ***AdeptVision Reference Guide*** for details on the transformation matrix.

The more common transformations are defined by the following arrays:

Table 2-17. Common Transformations

scale		
x	0	0
0	x	0
0	0	1

translate		
1	0	dx
0	1	dy
0	0	1

rotate		
cosθ	–sinθ	0
sinθ	cosθ	0
0	0	1

Examples

This example doubles the size of all graphics:

```
FOR i = 0 to 2     ;Define 0 elements
  FOR j = 0 to 2
    ta[i,j] = 0
  END
END

ta[0,0] = 2        ;Define scaling elements
ta[1,1] = 2
ta[2,2] = 1

GTRANS (20) ta[,]  ;Assume window is attached

; Graphics instructions go here

GTRANS (20)        ;Cancel transformation
```

The following example applies the basic camera calibration transformation to all subsequent graphics instructions. Graphics output will be placed based on the size of the field of view. In this example, the field of view is 30mm x 20mm, and an X is placed in the middle of the vision window:

```
ATTACH (glun, 4) "GRAPHICS"
FOPEN (glun) "Vision /MAXSIZE 640 480"
GTRANS (glun,1)
GTYPE (glun) 15, 10, "X"
```

Syntax

```
GTYPE (lun, mode) x, y, $text, font_numb, path, rotation
```

Function

Display a text string in a graphics window.

Usage Considerations

This instruction is available only with an graphics-based system.

Parameters

lun	Real value, variable, or expression (interpreted as an integer) that defines the logical unit number of the window to be accessed. (See the ATTACH instruction for a description of unit numbers.)
mode	Optional real value, variable, or expression (interpreted as an integer). Bits 1 and 2 of the value control attributes of the output. Bit 1 controls the relationship between the x, y parameters and the origin of the text string. Bit 2 controls the overstrike feature.The default is 0.
x, y	Real values, variables, or expressions (interpreted as an integer) specifying the X and Y coordinates (in pixels) of either the left end of the baseline on which the characters are written (mode bit 1 = 0), or the top left corner of the text string (mode bit 1 = 1), as shown in **Chapter Figure 2-4.**.
$text	String value, variable, or expression specifying the characters to be output.
font_numb	Optional real value, variable, or expression (interpreted as an integer) specifying the number of the character font to be used. The standard font (number 1) is assumed if this parameter is omitted.
path	Not currently used.
rotation	Not currently used.

Details

This instruction displays a text string at the specified coordinates. The text is drawn with the current graphics foreground color (see GCOLOR) and logical operation (see GLOGICAL). The current opaque/transparent mode is used, but the current texture pattern is ignored (see GTEXTURE).

If `mode` bit 2 = 1, the overstrike is defeated (designated overstrike characters are output to their own space).[1] If mode bit 2 = 0, designated overstrike characters will be displayed over the previous character.

This instruction operates on the window that is currently open for the specified logical unit. If the window is open in nonbuffered mode, the instruction sends a request to the window manager and takes effect immediately. If the window is open in buffered mode, the effect will be displayed when the buffer fills or an FEMPTY instruction is executed.

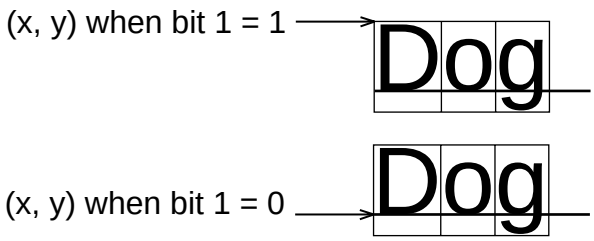


Figure 2-4. Effect of mode bit 1

[1] When a font is defined, characters can be designated as overstrike characters that will normally be printed on top of the previous character.

Examples

The following instruction draws, in the window currently open on graphics logical unit 20, the characters Label in font #1. The top left corner of the character string is at coordinates (10, 80):

```
GTYPE(20, 1) 10, 80, "Label", 1
```

The following instruction draws, in the window currently open on graphics logical unit 20, the characters Abcdef in the standard font. The left end of the baseline for the character string is at coordinates (50, 50).

```
GTYPE (20) 50, 50, "Abcdef"
```

Related Keywords

FOPEN (program instruction)

GCLIP (program instruction)

GCOLOR (program instruction)

GLOGICAL (program instruction)

GTEXTURE (program instruction)

GTRANS (program instruction)

Syntax

HALT

Function

Stop program execution and do not allow the program to be resumed.

Usage Considerations

The PROCEED command cannot be used to resume program execution after a HALT instruction causes the program to halt.

HALT forces an FCLOSE and/or DETACH on the disk and serial communication logical units as required.

Details

Causes a BREAK and then terminates execution of the application program regardless of any program loops remaining to be completed (see the EXECUTE command and instruction). The message (HALTED) is displayed.

After termination by a HALT instruction, program execution cannot be resumed with a PROCEED or RETRY command.

Related Keywords

PAUSE (program instruction)

RETURN (program instruction)

STOP (program instruction)

Syntax

HAND

Function

Return the current hand opening.

Usage Considerations

The HAND function returns information for the robot selected by the task executing the function.

If the V+ system is not configured to control a robot, use of the HAND function will not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

The word hand cannot be used as a program name or variable name.

Details

This function returns 0 if the hand is closed or 1 if the hand is opened or relaxed.

Related Keywords

CLOSE and **CLOSEI**	(program instructions)
OPEN and **OPENI**	(program instructions)
RELAX and **RELAXI**	(program instructions)
SELECT	(program instruction and real-valued function)

Syntax

```
... HAND.TIME
```

Function

Establish the duration of the motion delay that occurs during OPENI, CLOSEI, and RELAXI instructions.

Usage Considerations

The current value of the HAND.TIME parameter can be determined with the PARAMETER monitor command or real-valued function.

The value of the HAND.TIME parameter can be modified only with the PARAMETER monitor command or program instruction.

The parameter name can be abbreviated.

If the V+ system is controlling more than one robot, the HAND.TIME parameter controls the hand operation times for all the robots.

Details

The OPENI, CLOSEI, and RELAXI instructions are used to operate the hand after the robot has stopped moving. The HAND.TIME parameter determines the time allotted to the hand actuation before the next robot motion can be initiated.

The value for this parameter is interpreted as the number of seconds to delay. It can range from 0 to 10^{18}. Due to the way V+ generates time delays, the HAND.TIME parameter is internally rounded to the nearest multiple of 0.016 seconds.

This parameter is set to 0.05 seconds when the V+ system is initialized.

Example

```
PARAMETER HAND.TIME = 0.5
```

Sets the hand operation delay time to 0.5 seconds.

Related Keywords

CLOSEI (program instruction)

OPENI (program instruction)

RELAXI (program instruction)

PARAMETER (monitor command, program instruction, and real-valued function)

Syntax

```
HERE location_var
```

Function

Set the value of a transformation or precision-point variable equal to the current robot location.

Usage Considerations

The HERE instruction returns information for the robot selected by the task executing the instruction.

If the V+ system is not configured to control a robot, executing the HERE instruction will not generate an error due to the absence of a robot. However, the location value returned may not be meaningful.

The word here cannot be used as a program name or variable name.

Parameter

`location_var` Transformation, precision point, or compound transformation that ends with a transformation variable.

Details

This instruction sets the value of a transformation or precision-point variable equal to the current robot location.[1]

If the **`location_var`** is a compound transformation, only the right-most transformation is defined. An error message results if any of the other transformations in the compound transformation are not already defined.

[1] Normally, the robot location is determined by reading the instantaneous values of the joint encoders. However, if the robot has either backlash or linearity compensation enabled, the commanded robot location is used.

Examples

Set the transformation part equal to the current robot location:

```
HERE part
```

Assign the current location of the robot to the precision point #part:

```
HERE #part
```

Related Keywords

HERE (monitor command and transformation function)

SELECT (program instruction and real-valued function)

SET (program instruction)

Syntax

HERE

Function

Return a transformation value that represents the current location of the robot tool point.

Usage Considerations

The current location is obtained by reading the instantaneous value of the joint encoders so that it represents the actual location of the robot.[1]

The HERE function returns information for the robot selected by the task executing the function.

If the V+ system is not configured to control a robot, use of the HERE function does not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

The word here cannot be used as the name of a program or variable.

Example

Calculate the distance between the current robot location and the location the robot is currently moving to:

```
dist = DISTANCE(HERE, DEST)
```

Related Keyword

DEST (transformation function)

HERE (monitor command and program instruction)

SELECT (program instruction and real-valued function)

[1] If the robot has either backlash or linearity compensation enabled, this function returns the commanded robot location instead of the value determined from the joint encoder readings.

Syntax

HOUR.METER

Function

Return the current value of the robot hour meter.

Usage Considerations

This function applies only to the AdeptOne-MV, AdeptThree-MV, and Adept 550 robots.

You can type **listr hour.meter** to display the current value of the robot hour meter.

The word hour.meter cannot be used as a program name or variable name.

Details

The robot hour meter records the number of hours that Robot Power has been turned on. The meter is updated after one half hour of power-on time and hourly thereafter.

The hour meter is maintained by the Robot Signature Card (RSC) in the base of the robot. If the RSC does not respond, the HOUR.METER function returns the value –1.

Examples

To display the current reading of the robot hour meter on the system terminal, type

listr hour.meter

Assign the current reading of the robot hour meter to the real-valued variable start:

```
start = HOUR.METER
```

Syntax

```
ID (component, device, board)
```

Function

Return values that identify the configuration of the current system.

Parameters

component	Real value, variable, or expression (interpreted as an integer) whose value determines which component of identification information is returned.
device	Optional real value, variable, or expression (interpreted as an integer) whose value selects the device to be identified. Device #1 (the basic system) is assumed if this parameter is omitted.
board	Optional integer specifying the CPU of interest when device = 4. Board #1 (the main CPU) is assumed if this parameter is omitted.

Details

The ID function enables a program to access the information displayed by the ID monitor command. The values of the components are the same as the fields displayed by that command.

The function returns the value 0 for devices that do not exist.

Device number 1 refers to the **basic system**. The acceptable values for the **component** parameter, and the corresponding values returned, are:

component	Result of ID (component, 1)
1	Model designation of the system controller
2	Serial number of the system controller
3	Version number of the V+ software in use
4	Revision number of the V+ software in use
5	First option word for the V+ system (*)
6	Second option word for the V+ system (*)
7	Size of the system program memory (in kilobytes, K [1 K = 1024 8-bit bytes])

component	Result of ID (component, 1)
8	Controller option word (*)
9	Controller product-type value
10	(Value for internal use by Adept)
11	Controller hardware configuration (see below)

*The system and controller option words are described in **Appendix C**.

The value for the controller hardware configuration [returned by ID(11,1)] should be interpreted as a bit field, as follows:

Bit 1 (LSB) Graphics system processor (mask = 1)

This bit is 1 if the system processor in the controller includes hardware components for graphics output by the V+ system.

Bit 2 V+ monitor outputs to graphics monitor (mask = 2)

This bit is 1 if the system processor supports graphics output and the configuration switches on the system processor are set to direct the V+ monitor output to the graphics monitor. This bit is 0 if the V+ monitor output is directed to a terminal connected to the system controller.

Bit 3 Not used

Bit 4 SIO board present(mask = 8)

This bit is set if the SIO board is present in the controller.

Device number 2 refers to the manual control pendant. The **`component`** parameter must have the value 1. The value returned is the version number of the pendant software.

Device number 3 refers to the AdeptVision system, if one is included in the system. The acceptable values for the component parameter, and the corresponding values returned, are listed below. (If the system has multiple vision systems, the information returned is for the vision system that is currently selected.)

component	Result of ID (component, 3)
1	Model designation of the vision system
2	Serial number of the vision system
3	Version number of the vision software in use
4	Revision number of the vision software in use
5	Option word for the vision system
6	Size of the vision system RAM memory (in K)

Device number 4 refers to the system CPUs.

component	Result of ID (component, 4, board)
1	Number of the CPU
2	Not currently used
3	The CPU board version number
4	The CPU board revision number
5	CPU type: 1 = 68030; 0, 2 = 68040
6	Software modules active on the board: Bit field where bit 1 represents the V+ operating system, bit 2 represents the vision sofware, and bit 3 represents the servo/motion software
7	Size of the random access memory (RAM) on the CPU board (in K)

Device numbers 5 through 7 are invalid and return the value 0, as do any other specified devices that do not exist. For valid devices, an `*Invalid argument*` error message is reported if the requested component is not valid.

Device number 8 returns information for the currently selected robot. The acceptable values for the **`component`** parameter are the same as for a robot.

The values returned for device number *9* are for Adept's internal use.

Device number 10 refers to the external encoders connected to the robot controller. The acceptable values for the **`component`** parameter are the same as for a robot.

Device numbers 11, 12, ... refer to robot number 1, 2, ..., respectively, for each robot connected to the controller. That is, a device number equal to 10+*r* refers to robot number *r*, which can range from 1 to the value returned by the function SELECT(ROBOT, –1). The acceptable values for the **`component`** parameter, and the corresponding values returned, are listed below.

component	Result of ID (component, 10+*r*)
1	Model designation of the robot
2	Serial number of the robot
3	Number of motors configured for the robot. This normally is equal to the number of configured joints.
4	Value interpreted as bit flags for the robot joints that are enabled: bit 1 for joint 1, and so on. (The value is zero if the robot does not have joints that can be disabled selectively. For example, this value is defined for the X/Y/Z/Theta robot but is zero for the 4/5-axis SCARA module.)
5	Robot control-module identification number
6	Number for the robot control-module qualifier.
7	Number of robot joints configured for use
8	Robot option word (*)
9	Robot product-type value
10	Number of Cartesian axes used during motion planning. This value specifies how many values must be defined in the robot load-module arrays that specify the maximum Cartesian velocities and accelerations.
11	Second robot option word (*)
12	Information on the robot module. Currently only bit 1 (mask 1) is defined. If set, this bit indicates that the specified robot is an Adept robot.

*The system and controller option words are described in **Appendix C**.

Device number 50 refers to the currently selected force sensor. Device numbers 51 to 66 refer to force sensors numbered 1 to 16, respectively.

See the documentation for the SELECT program instruction for an explanation of selecting among multiple force sensors. The acceptable values for the **component** parameter, and the corresponding values returned, are listed below.

component	Result of ID (component, 50)
1	Model number of force sensor (0 if no force sensor is connected)
2	Serial number of force sensor (0 if no force sensor is connected)
3	Version number of VME Force Interface (VFI) firmware
4	Version number of force sensing software
5	Option word for force system
6	Size of the data collection buffer (in K)

Related Keywords

ID	(monitor command, see the *V+ Operating System Reference Guide)*
$ID	(string function)
SELECT	(monitor command, program instruction, and real-valued function)

Syntax

```
$ID (select)
```

Function

Return the system ID string.

Parameter

`select` Integer specifying the ID information to return. It may be:

–1 Return the system edit message.

–2 Return the edit letter for the V+ system.

Details

Return a string that identifies the release edition, date, and edit letter of the V+ system.

Syntax

```
IDENTICAL (location, location)
```

Function

Determine if two location values are exactly the same.

Parameter

location Transformation value that defines one of the locations of interest. This can be a function, a variable, or a compound transformation.

Details

This function returns the value TRUE if the positional and rotational components of the two specified locations are *exactly* the same. Even a single-bit difference in any of the components will result in the value FALSE being returned.

Example

The statement

```
x = IDENTICAL(base.1:loc,part)
```

will set the value of the real variable x to TRUE if the value of loc relative to the base.1 frame is exactly the same as the value stored in the variable part.

Related Keyword

DISTANCE (real-valued function)

Syntax

```
IF GOTO logical_expr label
```

Function

Branch to the specified label if the value of a logical expression is TRUE (nonzero).

Usage Considerations

In general, it is a better programming practice to use the IF ... THEN control structure rather than this instruction.

Parameters

`logical_expr`	Real-valued expression whose value is tested for TRUE (nonzero) or FALSE (zero).
`label`	Program step label number of a step in the current program.

Details

If the value of the expression is nonzero, program execution branches and begins executing the statement with a label matching the one specified. If the value of the expression is zero, the next instruction is executed as usual.

If the specified statement label is not defined, an error will occur if the branch is attempted.

Example

The most common use for IF...GOTO is as an exit-on-error instruction. The following code checks each I/O operation and branches to a label whenever an I/O error occurs:

```
ATTACH(dlun, 4) "DISK"
IF IOSTAT(dlun) < 0 GOTO 100

FOPENW(dlun) "my_file"
IF IOSTAT(dlun) < 0 GOTO 100

...

FCLOSE(dlun) "my_file"
IF IOSTAT(dlun) < 0 GOTO 100

DETACH(dlun)
IF IOSTAT(dlun) < 0 GOTO 100

100IF IOSTAT(dlun) < 0 THEN
   TYPE $ERROR(IOSTAT(dlun))
END
```

Related Keywords

GOTO (program instruction)

IF ... THEN (program instruction)

Syntax

```
IF logical_expr THEN

   first_steps
ELSE
   second_steps
END
```

Function

Conditionally execute a group of instructions (or one of two groups) depending on the result of a logical expression.

Usage Considerations

There must be a matching END statement for every IF... THEN in a program.

Parameters

`logical_expr` Real-valued expression whose value is tested for TRUE (nonzero) or FALSE (zero).

`first_steps` Optional group of program instructions that are executed only if the value of the logical expression is TRUE (nonzero).

`second_steps` Optional group of program instructions that are executed only if the value of the logical expression is FALSE (zero).

The ELSE statement may be omitted if there are no steps in this group.

Details

This control structure provides a means for conditionally executing one of two groups of instructions. In detail, it is processed as follows:

1. **`logical_expr`** is evaluated. If the result is FALSE (zero), skip to item 4 below.
2. The first group of instruction steps is executed.
3. Skip to item 5 below.
4. If there is an ELSE step, the second group of instruction steps is executed.
5. Program execution continues at the first step after the END step.

The ELSE and END steps must be on lines by themselves as shown.

There are no restrictions on the instructions that can be in either group in the structure. Thus, nested IF structures can be used.

Examples

Consider the following segment of a V+ program. If the value of row is greater than 5, the expression row > 5 will be TRUE (–1.0), so step 22 will be executed and 24 will not be executed. Otherwise, step 22 will not be executed, but step 24 will be executed:

```
21 IF row > 5 THEN
22   spacing = 10
23 ELSE
24   spacing = 20
25 END
```

The next program segment checks whether the variable input.signal has been defined. If it has, the program checks the signal indicated by the value of input.signal and types different messages depending on its setting. Note that the outer IF does not include an ELSE clause:

```
71 IF DEFINED(input.signal) THEN
72   IF SIG(input.signal) THEN
73     TYPE "The input signal is ON"
74   ELSE
75     TYPE "The input signal is OFF"
76   END
77 END
```

Refer to the **DEFINED** function for details on testing nonreal arguments.

Related Keywords

CASE (program instruction)

IF GOTO (program instruction)

Syntax

```
IGNORE signal
```

Function

Cancel the effect of a REACT or REACTI instruction.

Usage Considerations

Only digital I/O signals that are installed and configured as inputs are available for reaction monitoring.

The IGNORE instruction must be executed by the same program task that executed the REACT or REACTI instruction being canceled.

Parameter

signal Digital input signal number in the range 1001 to 1012, an internal signal in the range 2001 to 2008, or 0.

Details

Disables continuous monitoring of the specified signal, canceling the effect of the last REACT or REACTI for this signal.

This instruction has no effect if a value of zero is specified.

Example

```
IGNORE test
```

Stops monitoring of the digital input or soft signal identified by the value of test.

Related Keywords

LOCK (program instruction)

REACT (program instruction)

REACTI (program instruction)

Syntax

```
INRANGE (location)
```

Function

Return a value that indicates if a location can be reached by the robot and, if not, why not.

Usage Considerations

The INRANGE function returns information for the robot selected by the task executing the function.

Parameter

`location` Transformation function, variable, or compound that specifies a desired position and orientation for the robot tool tip.

Details

Returns system error bits that indicate whether the given location can be reached by the robot. A value of zero indicates the specified location can be reached.

If the location cannot be reached, the returned value is a coded binary number that identifies the error. A bit equal to 1 in the value indicates that the corresponding robot constraint has been violated as shown in the table below:

	Mask Value		
Bit #	**Hex**	**Decimal**	**Indication if bit set**
1	1	1	Joint 1 is limiting
2	2	2	Joint 2 is limiting
3	4	4	Joint 3 is limiting
4	8	8	Joint 4 is limiting
5	10	16	Joint 5 is limiting
6	20	32	Joint 6 is limiting
7	40	64	Joint 7 is limiting
8	80	128	Joint 8 is limiting
9	100	256	Joint 9 is limiting

	Mask Value		
Bit #	**Hex**	**Decimal**	**Indication if bit set**
10	200	512	Joint 10 is limiting
11	400	1024	Joint 11 is limiting
12	800	2048	Joint 12 is limiting
14	2000	8192	Location is too close in
15	4000	16384	Location is too far out
16	8000	32768	Joint vs. motor limiting, see below

If your motion system is configured to return motor as well as joint limit errors, bit 16 indicates whether a joint or motor is limiting the motion device move to *location*. If bit 16 is set, a motor is limiting. Otherwise, a joint is limiting. Since joints are checked first for limiting conditions, if bit 16 is set, all joints passed the limit checks and a motor is the limiting factor.

The mask values indicated above can be used with the BAND operator to determine if a corresponding bit is set.

Example

```
INRANGE(pallet:hole)
```

Returns a zero value if the robot can reach the location defined by the compound transformation pallet:hole.

If both joints 2 and 3 prevent the motion from being made, the value returned will be 6.

Related Keywords

BAND (operator)

SELECT (program instruction and real-valued function)

Syntax

```
INSTALL password, op
```

Function

Install or remove software options available to Adept systems.

Usage Considerations

You must have received the authorization password from Adept. INSTALL can only be run on CPU #1 in multiple CPU systems.

Parameters

password	A 15-character string assigned by Adept.
op	Optional integer indicating the desired operation: 0 = install option (default) 1 = remove option

Details

When you purchase additional software options from Adept, the software is delivered with a software license and authorization password that enables the software for a particular controller. If the option is not enabled, the software will not load correctly.

The password is keyed both to the software option and the serial number of your controller. The password cannot be used on any controller other than the one for which you purchased the software option.

> **NOTE:** If you replace the SIO module in your controller, you must reinstall the software options. Execute INSTALL for each system option after the new SIO has been installed.

Example

If you purchased the AIM MotionWare software from Adept and the password provided with the option is 4EX5-23GH8-AY3F, the following instruction will enable the software option:

```
INSTALL 4EX5-23GH8-AY3F
```

NOTE: Some options, such as AIM software, have additional software disks that must be copied to the hard drive. Other options, such as AdeptVision VXL, are already resident and need only to be enabled.

Syntax

INT (value)

Function

Return the integer part of the value.

Parameter

value Real-valued expression whose integer part is returned by this function.

Details

Returns the portion of the value parameter to the left of the decimal point (when the value is written without the use of scientific notation).

The value is not rounded before dropping the fraction.

The sign of the value parameter is preserved unless the result is zero.

Examples

```
INT(0.123)              ;Returns 0.0

INT(10.8)               ;Returns 10.0

INT(–5.462)             ;Returns –5.0

INT(1.3125E+2)          ;Returns 131.0

INT(cost+0.5)           ;Returns the value of "cost",
                        ;truncated to an integer.

INT(cost+0.5*SIGN(cost));Returns the value of "cost",
rounded
                        ;to the nearest integer. (The SIGN
                        ;function needs to be included to
                        ;correctly round negative values of
                        ;"cost".)
```

Related Keyword

FRACT (real-valued function)

Syntax

INTB ($string, first_char**)**

Function

Return the value of two bytes of a string interpreted as a signed 16-bit binary integer.

Parameters

$string String expression that contains the two bytes to be converted.

first_char Optional real-valued expression that specifies the position of the first of the two bytes in the string.

If first_char is omitted or has a value of 0 or 1, the first two bytes of the string are extracted. If first_char is greater than 1, it is interpreted as the character position for the first byte. For example, a value of 2 means that the second byte contains bits 9 to 16 and the third byte contains bits 1 to 8. An error is generated if first_char specifies a byte pair that is beyond the end of the input string.

Details

Two sequential bytes of a string are interpreted as being a 2's-complement 16-bit signed binary integer. The first byte contains bits 9 to 16, and the second byte contains bits 1 to 8.

The main use of this function is to convert binary numbers from an input data record to values that can be used internally by V+.

The expression

```
value = INTB($string, first_char)
```

is equivalent to the following instruction sequence:

```
value = ASC($string,first_char)*256 +
ASC($string,first_char+1)

IF value > 32767 THEN
   value = value-65536
END
```

To compute an unsigned integer, use: `INTB($string) BAND ^HFFFF.`

Examples

```
INTB($CHR(10)+$CHR(5))     ;Returns the value 2565
INTB($CHR(255)+$CHR(255))  ;Returns the value -1
```

Related Keywords

ASC	(real-valued function)
DBLB	(real-valued function)
FLTB	(real-valued function)
$INTB	(string function)
LNGB	(real-valued function)
VAL	(real-valued function)

Syntax

```
$INTB (value)
```

Function

Return a 2-byte string containing the binary representation of a 16-bit integer.

Parameter

value Real-valued expression, the value of which is converted to its binary representation.

Details

The integer part of a real value is converted into its binary representation and the low 16 bits of that binary representation are packed into a string as two 8-bit characters. Bits 9-16 are packed first, followed by bits 1-8.

This function is equivalent to:

```
$CHR(INT(value/256) BAND ^HFF) + $CHR(INT(value) BAND ^HFF)
```

The main use of this function is to convert integers to binary representation within an output record of a data file.

Example

```
$INTB(65*256+67) ;Returns the character string "AC".
```

Related Keywords

$CHR (string function)

$DBLB (real-valued function)

$FLTB (string function)

INTB (real-valued function)

$LNGB (real-valued function)

Syntax

```
... INTERACTIVE
```

Function

Control the display of message headers on the system terminal and requests for confirmation before performing certain operations.

Usage Considerations

The INTERACTIVE switch should be enabled for normal operation of V+ from the system terminal.

> **NOTE:** Disabling the INTERACTIVE switch is currently not recommended. INTERACTIVE is usually disabled only when the system is being controlled by a supervisory computer to relieve the computer from having to process the text of messages.

Details

When this switch is disabled, the following changes to system operation occur:

- V+ does not ask for confirmation before performing certain operations (for example, CALIBRATE).
- The text of error messages is not output.
- Headers on information messages are not output.
- Commands that normally produce continuous output (for example, STATUS, WHERE 1, and IO) display their output only once.
- The SWITCH monitor command displays information differently. Each requested switch is displayed as:

```
SWITCH_NAME [# elements displayed] [state of element
1]...[state of element n]
```

This switch is initially enabled.

Related Keywords

DISABLE	(monitor command and program instruction)
ENABLE	(monitor command and program instruction)
SWITCH	(monitor command, program instruction, and real-valued function)

Syntax

```
INT.EVENT source, level
```

Function

Send an event (as though from a SET.EVENT instruction) to the current task if an interrupt occurs on a specified VMEbus vector or a specified digital I/O signed transitions to positive.

Usage Considerations

VMEbus interrupts are available only on systems with the optional V+ Extensions license.

Parameters

source	Integer, expression, or real variable specifying: a VME bus vector in the range 192 - 255 a digital I/O signal in the range 1001 - 1003 the value 0, which cancels monitoring
level	Optional argument specifying the VME interrupt request level to be used. Acceptable values are 1 and 2.

Details

The CPU running the task must be configured for VME interrupts. Only two CPU boards within the system can be configured for interrupts.

If **source** is set to 0, signaling is canceled for the current task. When a task exits normally, is killed, or is reexecuted, signaling is canceled.

The V+ system can connect only one fast digital input to an interrupt at any one time. If you attempt to connect more than one, you get a `*Device not ready*` error message.

Example

```
INT.EVENT 201  ;Initiate monitoring of vector 201
   .
   .
   .
WAIT.EVENT     ;Wait until interrupt occurs
INT.EVENT 0    ;Cancel monitoring of interrupts
```

Related Keywords

GET.EVENT (program instruction)

SET.EVENT (program instruction)

WAIT.EVENT (program instruction)

Syntax

INVERSE (transformation)

Function

Return the transformation value that is the mathematical inverse of the given transformation value.

Parameter

transformation Transformation-valued expression.

Details

Mathematically, the value from this function is a transformation such that the value of the compound transformation shown below is the identity transformation (or NULL).

```
INVERSE(trans):trans
```

Stated another way, consider a transformation x that defines the location of object A relative to object B. Then INVERSE(x) is the transformation that defines the location of object B relative to A.

Example

Consider the case where the location part_1 is known in robot coordinates, and you want to find the location hole_1 with respect to part_1. We can use the compound expression:

```
part_1:hole_1
```

to represent the position of hole_1 in robot coordinates.

Suppose we move the robot to hole_1 and use the HERE command to define hole_pos as the position of hole_1 in robot coordinates. In other words, we want to find hole_1, knowing the values of part_1 and hole_pos, and knowing that:

```
part_1:hole_1is equal to hole_pos
```

We can then use the INVERSE function to determine hole_1 with the instruction:

```
SET hole_1 = INVERSE(part_1):hole_pos
```

Note that the SET instruction can be used without explicit use of INVERSE by using a compound transformation on the left-hand side, with identical results. That is, the instruction defines hole_1.

```
SET part_1:hole_1 = hole_pos
```

Syntax

```
IOGET_ (address, type, cpu)
```

Function

Return a value from global memory or from a device on the VME bus.

Usage Considerations

V+ does not enforce any memory protection schemes for global RAM or for accessing third-party boards. It is the programmer's responsibility to keep track of memory usage.

The forms of IOGET_ are:

IOGETB	Returns one unsigned byte
IOGETD	Returns 64-bit double-precision floating-point value
IOGETF	Returns 32-bit single-precision floating-point value
IOGETL	Returns one longword (32 bits)
IOGETW	Returns one unsigned word (16 bits)

Parameters

address	Integer representing the address to be referenced. The acceptable addresses are described below.
type	Optional integer specifying the memory space to be referenced:
0	CPU global RAM (default)
1	VME bus standard address space
2	VME bus short address space
cpu	Optional integer specifying the CPU to be accessed when type is 0. Acceptable values are 0 to the number of CPUs in the controller. The default is 0 (local CPU).

Details

The IOGET_ functions can access global memory on Adept CPUs and third-party boards. They can read only from a special section of CPU RAM and to the unprotected part of the VME bus address space. Local CPU memory and standard Adept devices cannot be accessed.

The CPU RAM accessible to the IOGET_ functions can be used for any desired purpose. For example, this RAM can be used as a global area shared by multiple processors. Note, however, that it is the programmer's responsibility to provide any interlocks needed to assure the integrity of shared memory (see IOTAS).

CPU global RAM 0 to ^H1FFF

The VME bus specification defines three independent address spaces: standard, short, and extended. Adept does not support the extended address space. The IOGET_ functions can access addresses in the following ranges:

Standard	0 to ^H3FFFFF (0 to ^H7FFFFF on nonvision systems). With a dual vision system, no user memory is available in the standard space.
Short	0 to ^H7FFF

CAUTION: All third-party boards used in Adept controllers must first be approved by Adept. Contact Adept applications support for details. Not all boards support the full range of addresses listed above.

Related Keywords

$IOGETS	(string function)
IOPUT_	(program instructions)
IOTAS	(real-valued function)

Syntax

```
$IOGETS (address, length, type, cpu)
```

Function

Return a string value from a device on the VME bus.

Usage Considerations

V+ does not enforce any memory protection schemes for global RAM or for accessing third-party boards. It is the programmer's responsibility to keep track of memory usage.

Parameters

address	Integer representing the address to be referenced. The acceptable addresses are shown below.
length	Integer specifying the length of the string to be read (1 - 128).
type	Optional integer specifying the memory space to be referenced:
0	CPU RAM (default)
1	VME bus standard address space
2	VME bus short address space
cpu	Optional integer specifying the CPU to be accessed when type is 0. Acceptable values are 0 to the number of CPUs in the controller. The default is 0 (local CPU).

Details

The $IOGETS function can access global memory on Adept CPUs and third-party boards. It can only write to a special section of CPU RAM and from the unprotected part of the VME bus address space. Local CPU memory and standard Adept devices cannot be accessed.

The CPU RAM accessible to the $IOGETS function can be used for any desired purpose. For example, this RAM can be used as a global area shared by multiple processors. Note, however, that it is the programmer's responsibility to provide any interlocks needed to assure the integrity of shared memory (see IOTAS).

CPU global RAM 0 to ^H1FFF

The VME bus specification defines three independent address spaces: standard, short, and extended. Adept does not support the extended address space. The IOGETS function can access addresses in the following ranges:

Standard	0 to ^H3FFFFF (0 to ^H7FFFFF on nonvision systems). With a dual vision system, no user memory is available in the standard space.
Short	0 to ^H7FFF

CAUTION: All third-party boards used in Adept controllers must first be approved by Adept. Contact Adept applications support for details. Not all boards support the full range of addresses listed earlier

Related Keywords

IOGET_	(real-valued functions)
IOPUT_	(program instruction)
IOTAS	(real-valued function)

Syntax

```
IOPUT_ address, type, cpu = value
```

Function

Write a value to global CPU memory or to a device on the VME bus.

Usage Considerations

V+ does not enforce any memory protection schemes for global RAM or for accessing third-party boards. It is the programmer's responsibility to keep track of memory usage.

The forms of IOPUT_ are:

IOPUTB	Writes one unsigned byte
IOPUTD	Writes 64-bit double-precision floating-point value
IOPUTF	Writes 32-bit single-precision floating-point value
IOPUTL	Writes one longword (32 bits)
IOPUTS	Writes a string value (length determined by string value)
IOPUTW	Writes one unsigned word (16 bits)

Parameters

address	Integer representing the address to be referenced. The acceptable values are described below.
type	Optional integer specifying the memory space to be referenced:
0	CPU RAM (default)
1	VME bus standard address space
2	VME bus short address space
cpu	Optional integer specifying the CPU to be accessed when type is 0. Acceptable values are 0 to the number of CPUs in the controller. The default is 0 (local CPU).
value	Value to write (must agree with type indicated by the instruction name—8-bit value for IOPUTB, string for IOPUTS, etc.).

Details

The IOPUT_ instructions can access global memory on Adept CPUs and third-party boards. They can write only to a special section of CPU RAM and to the unprotected part of the VME bus address space. Local CPU memory and standard Adept devices cannot be accessed.

The CPU RAM accessible to the IOPUT_ instructions can be used for any desired purpose. For example, this RAM can be used as a global area shared by multiple processors. Note, however, that it is the programmer's responsibility to provide any interlocks needed to assure the integrity of shared memory (see IOTAS).

CPU global RAM0 to ^H1FFF

The VME bus specification defines three independent address spaces: short, standard, and extended. Adept does not support the extended address space. The IOPUT_ instructions can access addresses in the following ranges:

Standard	0 to ^H3FFFFF (0 to ^H7FFFFF on nonvision systems). With a dual vision system, no user memory is available in the standard space.
Short	0 to ^H7FFF

CAUTION: All third-party boards used in Adept controllers must first be approved by Adept. Contact Adept applications support for details. Not all boards support the full range of address listed above.

Related Keywords

IOGET_	(real-valued functions)
$IOGETS	(string function)
IOTAS	(real-valued function)

Syntax

```
IOSTAT (lun, mode)
```

Function

Return status information for the last input/output operation for a device associated with a logical unit.

Usage Considerations

IOSTAT returns information only for the most recent operation. If more than one operation is performed, the status should be checked after each one.

Parameters

lun — Real-valued expression whose integer value is the logical unit number for the I/O device of interest. (See the description of ATTACH for information on the logical unit numbers recognized by the V+ system and how logical units are associated with I/O devices.)

mode — Optional expression that selects the type of I/O status to be returned for the specified logical unit. The following table shows the effects of the various mode values. (If the mode value is omitted, the value zero is assumed.)

Mode	Value returned by IOSTAT
0	Status of the last complete I/O operation
1	Status of a pending pre-read request
2	Size in bytes of the last file opened or of the last record read[a]
3	Status of any outstanding write request

[a] When sequential-access mode is being used, the byte count returned by IOSTAT(...,2) includes the carriage-return and line-feed characters at the end of each record.

Details

Unlike most V+ instructions, I/O instructions do not force the program to stop when an error is detected. Instead, the error status is stored internally for access with the IOSTAT function. This feature allows the program to interpret and possibly recover from many I/O errors.

When reading a file of unknown length, IOSTAT is the only method to determine when the end of the file is reached.

The value returned for modes 0, 1, and 3 is one of the following:

```
  1 = normal success
  0 = operation not yet complete
<0 = standard V+ error number
```

For `mode` = 3, the value 1 is also returned if no write request is outstanding.

See **Appendix B** for a description of standard V+ error numbers.

Examples

Try to open a file for reading, and make sure the file exists. If the file does exist, record its size (in bytes).

```
ATTACH (dlun, 4) "DISK"
FOPENR (dlun) "RECORD.DAT"
IF IOSTAT(dlun) < 0 THEN
   TYPE "Error opening file"
   HALT
END
file.size = IOSTAT(dlun,2)
```

Read and display records until the end of the file is reached.

```
ieeof = -504 ;End-of-file error code
READ (dlun) $record
WHILE IOSTAT(dlun) > 0 DO
   TYPE $record
   READ (dlun) $record
END
IF IOSTAT(dlun) == ieeof THEN
   TYPE "Normal end of file"
ELSE
   TYPE /B, "I/O error ", $ERROR(IOSTAT(dlun))
END
FCLOSE (dlun)
DETACH (dlun)
```

In the following example a TCP server program segment performs a no-wait read and then checks the status to determine if a client connection or disconnection was made.

```
ATTACH (lun,4) "TCP"
IF IOSTAT (lun) < 0 THEN
   TYPE "Attach error: ", $ERROR(IOSTAT(lun))
END

no_wait = 1
READ (lun, handle, no_wait) $in.str
status = IOSTAT(lun)

CASE status OF
  VALUE 100:                          ;New connection opened
   TYPE "New connection established.  Handle =", handle

  VALUE 101:                          ;Connection closed
   TYPE "Connection closed.  Handle =", handle

  VALUE –526:                         ;No data received
   WAIT

  ANY                                 ;Some other error
   TYPE "Error during READ:  ", $ERROR(status)
   GOTO 100
END
```

Related Keywords

ATTACH (program instruction)

FCLOSE (program instruction)

FCMND (program instruction)

FEMPTY (program instruction)

FOPEN_ (program instruction)

FSEEK (program instruction)

READ (program instruction)

WRITE (program instruction)

Syntax

IOTAS (address, type, cpu**)**

Function

Control access to shared devices on the VME bus.

Usage Considerations

V+ does not enforce any memory protection schemes for global RAM or for accessing third-party boards. It is the programmer's responsibility to keep track of memory usage.

Parameters

address	Integer representing the address to be referenced. The acceptable addresses are shown below.
type	Optional integer specifying the memory space to be referenced:
0	CPU global RAM (default)
1	VME bus standard address space
2	VME bus short address space
cpu	Optional integer specifying the CPU to be accessed when type is 0. Acceptable values are 0 to the number of CPUs in the controller. The default is 0 (local CPU).

Details

The IOTAS function can access global memory on Adept CPUs and third-party boards. It can access only a special section of CPU RAM and the unprotected part of the VME bus address space. Local CPU memory and standard Adept devices cannot be accessed.

This function sets the high-order (sign) bit of the byte at the specified address and returns the original status of the high-order bit. The returned values are:

0	Undefined memory location
1	Positive (sign bit was clear)
–1	Negative (sign bit was set)

This test-and-set operation is indivisible, which means that no other processor can modify the byte between the time it is tested and the time it is set.

The IOTAS function performs a hardware-level read-modify-write (RMW) cycle on the VME bus to make the test-and-set operation indivisible in a multiprocessing environment. If multiple processors all access this byte by using IOTAS, the byte can serve as an interlock between processors.

The CPU RAM accessible to the IOTAS function can be used for any desired purpose. For example, this RAM can be used as a global area shared by multiple processors. Note, however, that it is the programmer's responsibility to provide any interlocks needed to assure the integrity of shared memory.

CPU global RAM 0 to ^H1FFF

The VME bus specification defines three independent address spaces: short, standard, and extended. Adept does not support the extended address space. The IOPUT_ instructions can access memory addresses in the following ranges:

Standard	0 to ^H3FFFFF (0 to ^H7FFFFF on nonvision systems). With a dual vision system, no user memory is available in the standard space.
Short	0 to ^H7FFF

CAUTION: All third-party boards used in Adept controllers must first be approved by Adept. Contact Adept applications support for details. Not all boards will support the full range of address listed above.

Example

A common use of such an interlock is to allow, at most, one processor at a time to gain access to a data structure.

```
; Wait until the byte at ^H1000 on CPU 1 is not set

        WHILE IOTAS(^H1000, 0, 1) == –1 DO
       WAIT
   END

; Perform critical operation which requires interlocking

; Release the byte so another CPU can interlock it.

   IOPUTB(^H1000, 0, 1) = 0
```

If all processors use this means of interlocking, only one at a time may perform the critical operation.

Related Keyword

IOGET_	(real-valued functions)
$IOGETS	(string function)
IOPUT_	(program instructions)
TAS	(real-valued function)

Syntax

SPEED VALUE IPS *ALWAYS*

Function

Specify the units for a SPEED instruction as inches per second.

Usage Considerations

IPS can be used only as a parameter for a SPEED program instruction.

The speed setting specified is scaled by the monitor speed in effect when the robot motion occurs.

Speeds specified with the IPS parameter apply to straight-line motions. Joint-interpolated motions will not maintain the specified tool speed.

Details

This is an optional parameter for the SPEED program instruction, which specifies the units to be used for the speed value. That is, when IPS is specified in a SPEED instruction, the speed value is interpreted as inches/second (for straight-line motions).

See the description of the SPEED program instruction for further details on setting motion speeds with the IPS conversion factor.

Example

Set the robot tool tip speed to 20 inches/second for the next straight-line robot motion (assuming the monitor speed is set to 100):

```
SPEED 20 IPS
```

Related Keywords

MMPS (conversion factor)

SPEED (program instruction)

Syntax

```
... KERMIT.RETRY
```

Function

Establish the maximum number of times the (local) Kermit driver should retry an operation before reporting an error.

Usage Considerations

The current value of the KERMIT.RETRY parameter can be determined with the PARAMETER monitor command or real-valued function.

The value of the KERMIT.RETRY parameter can be modified only with the PARAMETER monitor command or program instruction.

The parameter name can be abbreviated.

Details

The KERMIT.RETRY parameter determines the maximum number of times the V+ driver for the Kermit protocol will retry an operation when a communication error occurs.

Examples of transmission errors are time-out errors and checksum errors indicating bad packets.

> **NOTE:** The setting of this parameter will not have the desired effect if the remote Kermit server has a lower retry threshold.

If long pauses are expected between successive read or write operations, the setting of this parameter should be increased. (The setting of the KERMIT.TIMEOUT parameter should also be increased in this case.)

The value for this parameter is interpreted as an integer value, which can range from 1 to 1000, inclusive.

This parameter is set to 15 when the V+ system is initialized.

Example

```
PARAMETER KERMIT.RETRY = 5 ;Set the retry limit to 5.
```

Related Keywords

KERMIT.TIMEOUT	(system parameter)
PARAMETER	(monitor command, program instruction, and real-valued function)

Syntax

```
... KERMIT.TIMEOUT
```

Function

Establish the delay parameter that the V+ driver for the Kermit protocol will send to the remote server.

Usage Considerations

The current value of the KERMIT.TIMEOUT parameter can be determined with the PARAMETER monitor command or real-valued function.

The value of the KERMIT.TIMEOUT parameter can be modified only with the PARAMETER monitor command or program instruction.

The parameter name can be abbreviated.

Details

This parameter has no direct effect on the (local) V+ driver for the Kermit communication protocol. The setting of this parameter is simply sent to the remote server to request it to wait up to the number of seconds specified between the packets sent by the local (V+) driver.

The time-out value should be increased any time long pauses are expected between read or write operations to a file open over the Kermit line. This can occur, for example, when debugging a program that has a file open over the Kermit line. (The setting of the KERMIT.RETRY parameter should also be increased in such cases.)

The value for this parameter is interpreted as a number of seconds. It can range from 1 to 95, inclusive.

This parameter is set to 8 when the V+ system is initialized.

Example

```
;Set time-out limit to 15 seconds.
PARAMETER KERMIT.TIMEOUT = 15
```

Related Keywords

KERMIT.RETRY (system parameter)

PARAMETER (monitor command, program instruction, and real-valued function)

Syntax

```
KEYMODE first_key, last_key = mode, setting
```

Function

Set the behavior of a group of keys on the manual control pendant.

Usage Considerations

The pendant must be attached before KEYMODE can be processed.

Parameters

`first_key` Real-valued expression that defines the first key number in a set of keys to be affected.

`last_key` Optional real-valued expression that defines the last key number in a set of keys to be affected. If no value is provided, only one key is affected.

`mode` Real-valued expression that defines the key mode to be set for the specified set of keys. The mode must have one of the following values (the modes are described below):

0	Keyboard mode
1	Toggle mode
2	Level mode
3	Special mode
4	Mask mode

`setting` Optional real-valued expression, which applies only when toggle mode is selected. The value determines the initial setting that will apply to the programmed keys. A zero value (the default assumed if no value is provided) will cause the keys to be off; any other value will cause them to be on.

Details

The various key modes are described below. See the description of the PENDANT real-valued function for more information on interaction with the manual control pendant.

0 - Keyboard Mode Keys programmed in this mode function similar to a terminal keyboard. A program can use the function PENDANT(0) to request the number of the next key pressed. The program will then wait until one of the keys programmed in KEYBOARD MODE is pressed. The number of the key is returned. Type-ahead is not possible—the program will not see any keys that are pressed while there is no PENDANT(0) function pending.

1 - Toggle Mode The state of the key may be read back on the fly. When the user presses a key that is in this mode, the internal state maintained by V+ is toggled. Also, the LED on the key (if any) is toggled. The LED is on when the key's state is ON. The state of the key is available even when the pendant is not in USER mode, but only if the pendant is attached.

2 - Level Mode The key's current level is maintained by the pendant and may be read on the fly. If the pendant is not in USER mode, the level returned for the key is zero. The key's state is ON only when it is actually being held down. This is useful, for example, for cursor control. The value returned is not valid if the pendant is not attached.

Whenever a key is programmed in level mode, its repeat mode is turned off.

3 - Special Mode Special mode is not implemented. It should not be used.

4 - Mask Mode Disables pendant keys in the specified range . When `setting` = 0, the key(s) are reenabled. For instance:

```
KEYMODE 25 = 4,1
```

causes the Run/Hold key to be ignored, and

```
KEYMODE 25 = 0
```

causes the Run/Hold key to have effect again.

Attach/Detach Requirements

The pendant must be attached (with the ATTACH program instruction) before the program can read keys using the PENDANT function, set the modes of any of the keys, or send text to the display.

Additionally, the pendant must be in USER mode for most input and output operations. The USER LED on the pendant lights when the pendant is in that mode. The LED blinks when a program request is pending because the pendant is not in USER mode.

Defaults

The key modes default to keyboard mode when the pendant is attached.

See the ***V+ Language User's Guide*** for details on programming the MCP.

Example

```
;Set the manual control soft keys to level mode.
KEYMODE 1,5 = 2
```

Related Keywords

ATTACH (program instruction)

PENDANT (real-valued function)

Syntax

```
KILL task_number
```

Function

Clear a program execution stack and detach any I/O devices that are attached.

Usage Considerations

KILL cannot be used while the specified program task is executing.

KILL has no effect if the specified task execution stack is empty.

Parameter

`task_number` Optional real value, variable, or expression (interpreted as an integer) that specifies which program task is to be cleared. (See below for the default. See the *V+ Language User's Guide* for information on tasks.)

Details

This operation clears the selected program execution stack, closes any open files, and detaches any I/O devices that may have been left attached by abnormal program termination.

This situation can occur if a program executes a PAUSE instruction or is terminated by an ABORT command or instruction, or an error condition, while an I/O device is attached or a file is open. If a limited-access I/O device (such as the serial I/O device) is left attached, no other program task can use that device until it is detached.

If the task number is not specified, the KILL command accesses task number 0 if the system is not in DEBUG mode; the current debug task is assumed if the system is in DEBUG mode. The KILL instruction always accesses task #0 if the task number is omitted.

Related Keywords

ABORT (monitor command and program instruction)

EXECUTE (monitor command and program instruction)

STATUS (monitor command, see the *V+ Operating System Reference Guide)*

Syntax

```
LAST (array_name[])
```

Function

Return the highest index used for an array (dimension).

Usage Considerations

If an automatic variable is referenced (see the AUTO instruction), this function returns the highest index *allocated* for the array, regardless of which elements have been assigned values.

Parameter

`array_name[]` Name of the array to be tested. Any type of V+ array variable can be specified: real-value, location, string, or belt. At least one array index must be omitted (see below).

Details

This function can be used to determine which elements of an array have already been defined. For one-dimension arrays (for example, part[ ]), this function returns the largest array index for which an element is defined. (See the first example below.)

For multiple-dimension arrays (for example, $names[,]), this function returns the largest array index for which an element is defined for the (left-most) dimension that is omitted from the array specification. (See the second example below.) There cannot be an index specified to the right of an omitted index.

Note that the value returned by this function is an index, not an array element. Furthermore, the value is not a count of the array elements that are defined—it is the largest index for which an array element is defined.

The value –1 is returned if the array does not have any elements defined for the requested dimension. That is, –1 is returned if any of the following situations occur:

- The array does not exist.
- The array has more or fewer dimensions than the number indicated in the function call. (For example, LAST(a[ ]) will return –1 if the array **a** has two dimensions.)
- The specified dimension in a multiple-dimension array has not been defined at all. (For example, LAST(a[20,]) returns –1 if LAST(a[,]) returns 19. That is, no elements a[20,i] exist.)

The error *Illegal array index* results if there is not at least one blank index in the array specification supplied to this function, or if there is an index specified to the right of a blank index.

Examples

```
LAST(part[])
```

If the array part[ ] has all its elements defined from part[0] through part[10], this example returns the value 10 (not 11, the number of elements defined).

```
LAST($names[2,])
```

If the given two-dimension array has elements [2,0], [2,3], and [2,5] defined, this example returns the value 5 (regardless of the status of elements [i,j] for i other than 2).

Syntax

```
LATCH (select)
```

Function

Return a transformation value representing the location of the robot at the occurrence of the last external trigger or AdeptForce guarded-mode trigger.

Usage Considerations

LATCH(0) returns information for the robot selected by the task executing the function. If the V+ system is not configured to control a robot, use of the LATCH(0) function will not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

LATCH(1) requires the AdeptForce VME option.

Parameter

`select` Optional integer, expression, or real variable specifying:

0 Robot postion latch (default)
1 AdeptForce guarded-mode trigger

Details

LATCH(0) returns a transformation value that represents the location of the robot when the last external trigger occurred. The LATCHED real-valued function should be used to determined when an external trigger has occurred and a valid location has been recorded.

Operation of the external trigger can be configured with the Adept controller configuration program (in the file CONFIG_C.V2 on the Utility Disk). This trigger may originate from the vision processor or an external source.

LATCH(1) returns the location of the robot at the last AdeptForce guarded-mode trigger. The LATCHED(1) real-valued function can be used to determine if an AdeptForce guarded-mode trigger has occurred.

See the ***AdeptForce VME User's Guide*** for details of the AdeptForce option.

Related Keywords

LATCHED (real-valued function)

#PLATCH (precision-point function)

Syntax

LATCHED (select)

Function

Return the status of the external trigger and/or an AdeptForce guarded-mode trigger.

Parameter

`select` Integer, expression, or real variable specifying:

0	Returns TRUE if the position of the currently selected robot was latched since the last time the function LATCHED(0) was executed. (Default)
–n	Where n refers to belt n, returns TRUE if the external encoder positions were latched since the last time the function LATCHED(–1) was executed.
1	Returns the number of the last AdeptForce guarded-mode trip condition trigger. If trip condition 1 has triggered, the value 1 is returned; if trip condition 2 has triggered, the value 2 is returned. If a guarded-mode trigger has not occurred since the last time the LATCHED(1) function was used, the value 0 is returned.

Details

This function returns a nonzero value if an external or guarded-mode trigger has occurred and the robot location or encoders have been latched since the function was last used. Otherwise, the function returns the value FALSE. When this function returns a nonzero value, a robot location and/or valid external encoder position(s) have been recorded due to the occurrence of the trigger.

> **NOTE:** After a nonzero value is returned by this function, subsequent uses of the function will return the value FALSE until the next occurrence of the trigger.

When this functions returns a nonzero value, the following functions can be used to access the latched information:

DEVICE	Returns position of external encoder
LATCH	Returns robot location as a transformation
#PLATCH	Returns robot location as a precision point

Operation of the external trigger can be configured with the Adept controller configuration program (in the file CONFIG_C.V2 on the Utility Disk). This trigger may originate from the vision processor or an external source.

For details of the AdeptForce option, see the ***AdeptForce VME User's Guide***. For hardware details about LATCH and TRIGGER, see the ***Adept MV Controller User's Guide***.

Related Keywords

DEVICE	(real-valued function)
LATCH	(transformation function)
#PLATCH	(precision-point function)

Syntax

LEFTY

Function

Request a change in the robot configuration during the next motion so that the first two links of a SCARA robot resemble a human's left arm.

Usage Considerations

Configuration changes cannot be made during straight-line motions.

If the selected robot does not support a left-handed configuration, this instruction is ignored by the robot.

The LEFTY instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the LEFTY instruction will cause an error.

Figure 2-5. shows the LEFTY/RIGHTY configurations (top view of robot).

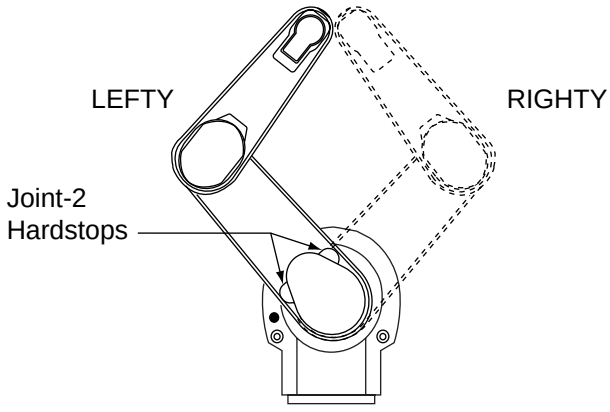


Figure 2-5. LEFTY/RIGHTY

Related Keywords

CONFIG	(real-valued function)
RIGHTY	(program instruction)
SELECT	(program instruction and real-valued function)

Syntax

```
LEN (string)
```

Function

Return the number of characters in the given string.

Parameter

string	String constant, variable, or expression whose length is to be computed.

Example

Return the number of characters in the string $str:

```
$str = "Hello"
str.len = LEN($str)
```

Syntax

LNGB ($string, first_char**)**

Function

Return the value of four bytes of a string interpreted as a signed 32-bit binary integer.

Usage Considerations

Since single-precision numbers are stored internally with only 24 bits of significance, input values that contain more than 24 significant bits will be converted with some loss in precision.

Double-precision numbers are stored with 32 bits of significance with the MSB being the sign bit. Doubles are converted with no loss of precision.

Parameters

$string String constant, variable, or expression that contains the four bytes to be converted.

first_char Optional real value, variable, or expression (interpreted as an integer) that specifies the position of the first of the four bytes in the string. An error results if first_char specifies a series of four bytes that goes beyond the end of the input string.

If first_char is omitted or has the value 0 or 1, the first four bytes of the string are extracted. If first_char is greater than 1, it is interpreted as the character position for the first byte (see below).

Details

Four sequential characters (bytes) of a string are interpreted as being a 2's-complement 32-bit signed binary integer. The first of the four bytes contains bits 25 to 32 of the integer, the second of the four bytes contains bits 17 to 24, etc.

For example, if `first_char` has the value 9, then the ninth character (byte) in the input string contains bits 25 to 32 of the integer, the tenth byte of the string contains bits 17 to 24, and so forth.

The main use of this function is to convert binary numbers from an input data record to values that can be used internally by V+.

Example

```
LNGB($INTB(1)+$INTB(5)) ;Returns the value 65541
```

Related Keywords

ASC	(real-valued function)
DBLB	(real-valued function)
FLTB	(real-valued function)
INTB	(real-valued function)
$LNGB	(string function)
TRANSB	(real-valued function)
VAL	(real-valued function)

Syntax

```
$LNGB (value)
```

Function

Return a 4-byte string containing the binary representation of a 32-bit integer.

Parameter

value	Real value, variable, or expression whose value is to be converted to its binary representation.

Details

The integer part of a real value is converted into its binary representation; the low 32-bits of that binary representation are packed into a string as four 8-bit characters. Bits 25 to 32 are packed into the first byte, followed by bits 17 to 24 in the second byte, and so forth.

The main use of this function is to convert integer values to binary representation within an output record of a data file.

The operation performed by this function is equivalent to the following expression:

```
$CHR(INT(value/^H1000000) BAND ^HFF)
+ $CHR(INT(value/^H10000) BAND ^HFF)
+ $CHR(INT(value/^H100) BAND ^HFF)
+ $CHR(INT(value) BAND ^HFF)
```

Example

```
$LNGB(67*65536+12345) ;Returns the value
$INTB(67)+$INTB(12345)
```

Related Keywords

$CHR	(string function)
$FLTB	(string function)
$INTB	(string function)
LNGB	(real-valued function)
$TRANSB	(string function)

Syntax

```
LOCAL type variable, ..., variable
```

Function

Declare permanent variables that are defined only within the current program.

Usage Considerations

Subroutines can be called simultaneously by multiple program tasks and recursively by a single task. Local and global variables could be corrupted if such calls occur inadvertently. Thus, the use of automatic variables in place of local variables is recommended.

LOCAL statements must appear before any executable statement in the program—only the .PROGRAM statement, comments, blank lines, AUTO statements, and other LOCAL statements may precede a LOCAL statement.

If a variable is listed in a LOCAL statement, any global variable with the same name cannot be accessed directly by that program.

The values of local variables are not saved (or restored) by the STORE (or LOAD) monitor command.

Parameters

`type`	Optional parameter specifying the type of a variable. The acceptable types are:
LOC	Location variable (transformation or precision point)
REAL	Single-precision real variable
DOUBLE	Double-precision real variable
	See the description of the GLOBAL program instruction for details on the default type.
`variable`	Variable name (belt, precision point, real-value, string, or transformation). Each variable can be a simple variable or an array. Array variables must *not* have their indexes specified. If a `type` is specified, all variables must be of that type.

Details

This instruction is used to declare variables to be defined only within the current program. That is, a local variable can be referenced only within its own program. Also, the names of local variables can be selected without regard for the names of local variables defined in other programs.

Local variables are allocated only once during program execution, and their values are preserved between successive subroutine calls. These values are also shared if the same program is executed by multiple program tasks.

If a program that uses LOCAL (or global) variables is called by several different program tasks, or called recursively by a single task, the values of those variables can be modified by the different program instances and cause very strange program errors. Therefore, automatic variables should be used for all temporary local variables to minimize the chance of errors. (See the AUTO instruction.)

Variables can be defined as automatic, global, or local. Once a variable has been assigned to a class, an attempt to assign the variable to a different class will result in the error *Attempt to redefine variable class*.

Variables can be defined only once within the same context (automatic, local, or global). Attempting to define a variable more than once (that is, with a different type) will yield the error ∗Attempt to redefine variable type∗.[1]

Local variables can be referenced with monitor commands such as BPT, DELETE_, DO, HERE, LIST_, POINT, TEACH, TOOL, and WATCH by using the optional context specifier @. The general syntax is:

```
command @task:program command_arguments
```

See the ***V+ Language User's Guide*** for more information on specifying program context.

Example

Declare the variables loc.a, $ans, and i to be local to the current program:

```
LOCAL loc.a, $ans, i
```

Related Keywords

AUTO (program instruction)

GLOBAL (program instruction)

[1] See the chapter Data Types and Operators in the ***V+ Language User's Guide***.

Syntax

```
LOCK priority
```

Function

Set the program reaction lock-out priority to the value given.

Usage Considerations

LOCK 0 is assumed whenever program execution is initiated and when a new execution cycle begins.

Changing the priority may affect how reactions will be processed. Before using this instruction, be sure you know what reactions are active (and their priorities).

Parameter

`priority` Real-valued expression with a value from 0 to 127, which will become the new reaction lock-out priority.

Details

When a program is EXECUTEd, it is placed on the execution stack. When the program's task becomes the highest priority task in a time slice, the program's priority is set to 0 and it begins execution. During actual execution, a program's task can be suspended at the end of a time slice, in which case the task waits until the next time it is the highest priority task in a time slice. The LOCK instruction does not affect the task priority value within a time slice: It only changes the program priority of an executing program.

Program priority becomes important when a reaction routine (REACT, REACTE, REACTI) is invoked. A program can defer execution of a REACT or REACTI routine by setting the temporary program priority to a value higher than the REACT or REACTI program priority. This is the function of a LOCK instruction. For example, if a LOCK instruction changes the temporary program priority to 20, any REACT or REACTI interrupts with lower priority values are deferred. (REACTE routines cannot be deferred by priority considerations.)

Deferred reactions are not ignored. Every time a new LOCK instruction is processed, any deferred reaction programs are checked to see if their priority is high enough for them to execute. As soon as the program priority is lowered, all pending reaction routines with a higher priority are run according to their relative priority.

The PRIORITY real-valued function can be used to determine the program priority at any time.

> **NOTE:** Although a LOCK instruction can be used to change the program priority within a reaction program, the priority will still be returned to its prereaction value when a RETURN is executed in the program. This occurs only when executing a RETURN from a *reaction* program.

Example

```
LOCK PRIORITY+10 ;Increase the program priority by 10.
```

Related Keywords

PRIORITY (real-valued function)

REACT (program instruction)

REACTI (program instruction)

Syntax

```
MAX (value, ..., value)
```

Function

Return the maximum value contained in the list of values.

Parameter

value Each value in the list can be specified as a real-valued constant, variable, or expression.

Details

The list of values provided is scanned for the largest value, and that value is returned by the function.

The sign of each value is considered. Thus, for example, the value –10 is considered larger than –100.

Example

The program instruction:

```
max.value = MAX(x, y, z, 0)
```

sets **max.value** to the largest value of the variables **x**, **y**, and **z**, or to zero if all three variables have values less than zero.

Related Keyword

MIN (real-valued function)

Syntax

```
MC monitor_command
```

Function

Introduce a monitor command within a command program.

Usage Considerations

The MC instruction can be contained only within a command program. (Command programs can contain *only* MC instructions, blank lines, and comment lines.)

Parameter

`monitor_command` Any valid V+ monitor command.

Details

Command programs are created using one of the V+ editors. To indicate to the editor that a command program, rather than a normal program, is being created. Every operation line of a command program must begin with the letters MC (that is, for Monitor Command follows) followed by one or more spaces. As with regular application programs, command programs can contain blank lines and comment lines to add clarity.

Every nonblank line of a command program must contain a monitor command (or a comment). Monitor commands and program instructions cannot be mixed. Program instructions can be included, however, by using the DO command. That is, to include an instruction in a command program, you can type a line with the form **mc do** instruction. See the ***V+ Operating System Reference Guide*** for details on monitor commands.

Example

The following command program loads disk files, prepares for execution of a program, and begins the execution. Note that a DO command is used to include a MOVE instruction:

```
1 .PROGRAM setup()
2  MC LOAD C:project
3  MC LOAD B:project.lc
4  MC SPEED 50
5  MC DO MOVE safe.loc
6  MC EXECUTE motion, -1
7 .END
```

Related Keywords

COMMANDS	(monitor command, see the *V+ Operating System Reference Guide)*
MCS	(program instruction)

Syntax

```
... MCP.MESSAGE
```

Function

Conrol how system error messages are handled when the controller keyswitch is *not* in the MANUAL position.

Usage Considerations

Systems without an external front panel assume that the keyswitch in the AUTO position.

Details

If this switch is enabled, all system error messages are output to the manual control pendant regardless of the setting of the controller keyswitch. In that case, the system operator must press CLR ERR on the manual control pendant to clear the error condition before system operation can continue.

By default, this switch is disabled. That is, error messages are output only to the device selected by the controller keyswitch.

Related Keywords

MCS.MESSAGE (system switch)

DISABLE (monitor command and program instruction)

ENABLE (monitor command and program instruction)

SWITCH (monitor command, program instruction, and real function)

Syntax

```
MCS string
```

Function

Invoke a monitor command from an application program.

Parameter

`string` String value, variable, or expression that defines one of the V+ monitor commands listed below.

Details

Normally, monitor commands can be invoked only from the system terminal or from command programs (which contain only monitor commands). The MCS instruction can be used to invoke the following monitor commands from an application program:

DELETE	DELETEL	DELETEP	DELETER	DELETES
FCOPY	LOAD	STORE	STOREL	STOREM
STOREP	STORER	STORES	VRENAME	

Using these commands, an application program can store, load, and copy programs to and from disk, and also delete programs from memory to make room for other programs. Similarly, variables can be deleted from memory when they are no longer needed. Also, vision prototypes can be renamed.

> **NOTE:** If the monitor command specified in the string parameter contains a blank program context (that is, it contains @), any variables listed in the command will be treated as though they are referenced within the program containing the MCS instruction. (See the *V+ Language User's Guide* for more information on program context.)

Program execution will not be stopped if an error occurs while processing the monitor command. The ERROR real-valued function can be used after the MCS instruction to check for the occurrence of an error.

> **NOTE:** If a DELETE_ command is used within a subroutine to delete one of the subroutine parameters (that is, one of the variables in the .PROGRAM statement), the variable will not be deleted and no error condition will be recorded.

Normal output by the monitor command to the system terminal will be done if the MCS.MESSAGE system switch is enabled. For example, the LOAD command would output the .PROGRAM lines from each program loaded. (The MCS.MESSAGE switch is normally disabled.)

If the FCOPY option is used, logical units 5 (disk #1) and 6 (disk #2) must be available. If LOAD or STORE_ are used, logical unit # 5 must be available.

Example

The following program loads a disk file, executes the program in the file, and deletes the program from the system memory. Then another program file is loaded into memory and executed. (Although this simple example could also be implemented with a command program, the following demonstrates use of the MCS instruction in a normal program.)

```
.PROGRAM admin()
   MCS "LOAD C:setup"
   CALL setup
   MCS "DELETEP setup"
   MCS "LOAD C:demo_1"
   CALL demo_main
.END
```

Related Keywords

MC (program instruction)

MCS.MESSAGE (system switch)

Syntax

```
... MCS.MESSAGE
```

Function

Enable or disable output to the system terminal from monitor commands executed with the MCS instruction.

Details

If this switch is enabled, output from monitor commands executed with the MCS instruction is displayed on the system terminal. Otherwise, output is suppressed.

For example, when this switch is disabled, a LOAD command invoked with the MCS instruction does not output .PROGRAM information lines or error messages.

By default, this switch is disabled, suppressing output from commands invoked with the MCS instruction.

Related Keywords

DISABLE (monitor command and program instruction)

ENABLE (monitor command and program instruction)

MCP.MESSAGE (system switch)

MCS (program instruction)

MESSAGES (system switch)

SWITCH (monitor command, program instruction, and real function)

Syntax

```
... MESSAGES
```

Function

Enable or disable output to the system terminal from TYPE instructions.

Details

If this switch is enabled, output from TYPE instructions is displayed on the system terminal. Otherwise, output is suppressed.

By default, this switch is enabled, allowing output to occur.

Related Keywords

DISABLE (monitor command and program instruction)

ENABLE (monitor command and program instruction)

MCS.MESSAGE (system switch)

SWITCH (monitor command, program instruction, and real-valued function)

TYPE (program instruction)

Syntax

```
$MID (string, first_char, num_chars)
```

Function

Return a substring of the specified string.

Parameters

string	String variable, constant, or expression from which the substring is to be extracted.
first_char	Optional real-valued expression that specifies the first character of the substring.
num_chars	Real-valued expression that specifies the number of characters to be copied to the substring.

Details

If first_char is omitted or has a value less than or equal to 1, the substring starts with the first character of **string**. If first_char is larger than the length of the input string, the function will return an empty string.

If there are fewer than **num_chars** characters from the specified starting character position to the end of the input string, the output string will consist of only the characters up to the end of the input string. That is, no error results and the output string is not extended to the requested length.

Example

The instructions below result in the string variable $substring containing the string cd, since cd is the 2-character string that starts at character position 3 of the string abcde contained in the string variable $string:

```
$string = "abcdef"
$substring = $MID($string, 3, 2)
```

Related Keyword

$UNPACK (string function)

Syntax

```
MIN (value, ..., value)
```

Function

Return the minimum value contained in the list of values.

Parameter

value Each value in the list can be specified as a real-valued constant, variable, or expression.

Details

The list of values provided is scanned for the smallest value, and that value is returned by the function.

The sign of each value is considered. Thus, for example, the value –100 is considered smaller than –10.

Example

The program instruction:

```
min.value = MIN(1000, x, y, z)
```

will set min.value to the smallest value of the variables x, y, and z, or to the value 1000 if all three variables have values greater than 1000.

Related Keyword

MAX (real-valued function)

Syntax

```
SPEED value MMPS  ALWAYS
```

Function

Specify the units for a SPEED instruction as millimeters per second.

Usage Considerations

MMPS can be used only as a parameter for a SPEED program instruction.

The speed setting specified is scaled by the monitor speed in effect when the robot motion occurs.

Speeds specified with the MMPS parameter apply to straight-line motions. Joint-interpolated motions will not maintain the specified tool speed.

Details

This is an optional parameter for the SPEED program instruction, which specifies the units to be used for the speed value. That is, when MMPS is specified in a SPEED instruction, the speed value is interpreted as millimeters/second (for straight-line motions).

See the description of the SPEED program instruction for further details on setting motion speeds with the IPS conversion factor.

Example

Set the default program speed for straight-line motions to 10 millimeters per second (assuming the monitor speed is set to 100):

```
SPEED 10 MMPS ALWAYS
```

Related Keywords

IPS (conversion factor)

SPEED (program instruction)

Syntax

```
... value MOD value ...
```

Function

Compute the modulus of two values.

Details

The MOD operator operates on two values, resulting in a value that is the **remainder** after dividing the first value by the second value. (The second value cannot be zero.)

See the *V+ Language User's Guide* for the order in which operators are evaluated within expressions.

Examples

```
5 MOD 2 ;Returns 1 (5/2 is 2 with a remainder of 1)

81 MOD 27 ;Returns 0 (81/27 is 3 with a remainder of 0)
```

Syntax

```
... MONITORS
```

Function

Enable or disable selecting of multiple monitor windows.

Usage Considerations

This switch is used with systems configured for multiple V+ system processors (requires the optional V+ Extensions software).

Details

When enabled, monitor windows for the auxiliary CPUs can be selected from the **adept** pull-down menu on the top-level menu bar. When disabled, windows for the auxiliary CPUs have their names dimmed and cannot be selected.

Default is OFF.

Syntax

```
MOVE location

MOVES location
```

Function

Initiate a robot motion to the position and orientation described by the given location.

Usage Considerations

MOVE causes a joint-interpolated motion.

MOVES causes a straight-line motion, during which no changes in configuration are permitted.

These instructions can be executed by any program task so long as the task has attached a robot. The instructions apply to the robot selected by the task.

If the V+ system is not configured to control a robot, executing these instructions will cause an error.

Parameter

`location`	Transformation, precision point, location function, or compound transformation that specifies the destination to which the robot is to move.

Details

The MOVE instruction causes a joint-interpolated motion. That is, intermediate set points between the initial and final robot locations are computed by interpolating between the initial and final joint positions. Any changes in configuration requested by the program (for example, by a LEFTY instruction) are executed during the motion.

The MOVES instruction causes a straight-line motion. During such a motion the tool is moved along a straight-line path and is smoothly rotated to its final orientation. No changes in configuration are allowed during straight-line motions.

Examples

MOVE #pick Move by joint-interpolated motion to the location described by the precision point #pick.

MOVES ref:place Move along a straight-line path to the location described by the compound transformation ref:place.

Related Keywords

APPRO and **APPROS** (program instructions)

DEPART and **DEPARTS** (program instructions)

MOVEF and **MOVESF** (program instructions)

MOVET and **MOVEST** (program instructions)

SELECT (program instruction and real-valued function)

Syntax

```
     MOVEF location, depart_clr, appro_clr, depart_tqe,
horiz_accel_tqe, horiz_decel_tqe, appro_tqe, model

     MOVESF location, depart_clr, appro_clr, depart_tqe,
horiz_accel_tqe, horiz_decel_tqe, appro_tqe, model
```

Function

Initiate a three-segment pick-and-place robot motion to the specified destination, moving the robot at the fastest allowable speed.

Usage Considerations

These instructions are available only if your system controls an Adept robot that supports optimized moves (version 11.1 or later).

MOVEF causes a joint-interpolated motion during the horizontal motion segment.

MOVESF causes a straight-line motion during the horizontal motion segment. For the entire motion, no changes in configuration are permitted.

Unlike the DEPART, DEPARTS, APPRO, and APPROS instructions, which move the robot along the *tool*-Z axis, the departure and approach motion segments initiated by these instructions move only joint 3 (the linear Z axis), and those motions are *independent* of the TOOL transformation.

Before these instructions can be executed, the SFUTIL utility program must be used to compute the coefficients for the dynamic model of the robot.

These instructions can be executed by any program task, so long as the task has attached a robot. The instructions apply to the robot selected by the task.

If the V$^+$ system is not configured to control a robot, an attempt to execute one of these instructions causes an error.

Parameters

`location` Transformation, precision point, location function, or compound transformation that specifies the destination to which the robot is to move.

`depart_clr` Optional real-valued expression that specifies the minimum clearance distance (in millimeters) that joint 3 is to be moved relative to its initial location during the first segment of the motion.

A positive distance retracts (lifts up) joint 3 from the initial location. A negative distance extends (drops down) joint 3 from the initial location. The distance zero eliminates the first segment of the pick and place motion.

Unlike the DEPART and DEPARTS instructions, which specify the full length of the vertical excursion, this parameter specifies the *minimum* desired clearance height before the horizontal motion is initiated. Based upon the dynamic characteristics of joint 3 and the horizontal motion, the V+ system automatically computes the total length of the departure motion. For many high-speed motions, the minimum clearance height should be set to a fraction (for example, 1/6th) of the distance that would normally be specified as the depart height.

`appro_clr` Optional real-valued expression that specifies the minimum clearance distance (in millimeters) that joint 3 is to be posed above the final destination at the time the horizontal motion is concluded.

A positive distance indicates that joint 3 should be posed above the final destination. A negative distance places joint 3 below its final destination. The distance zero eliminates the final segment of the pick and place motion so that the robot moves directly to the destination from the departure point.

As with the depart_clr parameter, the approach clearance height specifies the minimum clearance distance, rather than the total length of the vertical excursion.

`depart_tqe` Optional real-valued expression that specifies the percentage of maximum torque that should be utilized to accelerate joint 3 during the departure motion. This parameter must be specified if depart_clr is nonzero. However, this parameter is ignored if depart_clr is zero.

If this parameter is specified, its value must be in the range 30.0 to 100.0, where 100.0 indicates that all the available torque should be applied.

horiz_accel_tqe Real-valued expression that specifies the percentage of maximum torque that should be utilized to accelerate the robot during the start of the second (horizontal) motion segment.

During the second motion segment, all the joints of the robot are moved from their depart positions to their approach positions. Thus, in general, this parameter controls the torque applied to accelerate all the axes during the major segment of the motion. However, in many cases, the second motion segment is primarily a horizontal sweeping motion. Thus, typically, this parameter controls the maximum torque applied to accelerate joints 1 and 2 during the pick-and-place motion.

The value of this parameter must be in the range 30.0 to 100.0, where 100.0 indicates that all the available torque should be applied.

horiz_decel_tqe Real-valued expression that specifies the percentage of maximum torque that should be utilized to decelerate the robot during the end of the second (horizontal) motion segment. This parameter is the deceleration counterpart of the horiz_accel_tqe parameter.

The value of this parameter must be in the range 30.0 to 100.0, where 100.0 indicates that all the available torque should be applied.

appro_tqe Optional real-valued expression that specifies the percentage of maximum torque that should be utilized to decelerate joint 3 during the motion from the approach position to the final destination. This value must be specified if appro_clr is nonzero. However, this parameter is ignored if appro_clr is zero.

If this parameter is specified, its value must be in the range 30.0 to 100.0, where 100.0 indicates that all the available torque should be applied.

model Real-valued expression (interpreted as an integer) that specifies which set of dynamic coefficients should be used in

the dynamic model to predict the performance of the robot during the pick-and-place motion.

The dynamic model is evaluated to determine the fastest speed at which the robot can be moved to the destination without violating the specified torque limits. As with most mechanical systems, the values of the dynamic coefficients are a function of the robot tooling and payload. Therefore, a different set of coefficient values must be specified each time the tooling or payload changes. Up to six sets of dynamic coefficients can be stored per robot. Thus, the value of model can range from 1 to 6.

Details

The MOVEF and MOVESF instructions initiate the execution of a three-segment pick-and-place motion, and automatically compute the fastest speed, acceleration, and deceleration rates for each motion segment.

The three-segment motion is roughly equivalent to executing a DEPART motion, followed by an APPRO motion, followed by a final MOVE to the destination. For simplicity, the depart motion and the move from the approach point to the final destination affect only the position of the linear Z axis of the SCARA robot. However, the second segment, which moves from the depart position to the approach position, is a general joint-interpolated (for MOVEF) or straight-line motion (for MOVESF). Thus that segment can affect all the axes of the robot.

In order to automatically compute the fastest speed, acceleration, and deceleration rates at which to move the robot, these instructions analyze a dynamic model of the robot that is built into the V+ system. The model allows the system to predict the motor torques required to move the robot and to select the motion times that drive the robot in the shortest amount of time without violating the maximum torque specifications. For applications that are sensitive to speed, this allows application programs to be written that operate the robot at near optimum cycle times without the need of special, manual, speed tuning.

Although the dynamic model of the robot is fixed by the design of the robot, the coefficients for the model depend on the robot, its tooling, and its payload. In order to simplify the determination of the coefficient values, Adept provides a utility program (SFUTIL) to automatically compute the coefficient values. The utility program does this by moving the robot within a small section of the workspace and analyzing the resulting dynamic response. Thus, the SFUTIL program must be executed prior to using the MOVEF and MOVESF instructions, and it must be run once for each set of coefficients that is required.

Since the motion times are computed automatically by these instructions, the settings of the ACCEL and DURATION instructions are ignored during the pick-and-place motion. However, as a convenience during system setup, the SPEED monitor command and the SPEED program instruction can be used to scale down the motion speeds. If the combination of the monitor SPEED and the program SPEED is 100.0 or greater, the MOVEF and MOVESF instructions drive the robot to the maximum allowable torque levels (that is, as specified by the instruction parameters). However, if the combined speed is below 100.0, the duration of each of the motion segments is proportionally increased. While debugging an application, this allows the operator to see the exact motion that will be performed, but at a reduced speed. Note that unlike other multiple-segment paths generated by V+, reducing the value set by the SPEED program instruction does not cause the path generated by MOVEF or MOVESF instructions to be different from that followed at full speed.

Example

The instruction below generates a three-part motion to the destination pick. The robot will begin by retracting the Z axis 10 millimeters, using 99% of the torque available in motor 3. Upon reaching the depart clearance height, the robot will immediately start moving to an approach position 15 millimeters above pick. During this second segment, 100% of the motor torques will be applied to accelerate the robot. However, to reduce the deceleration impulse imparted to the robot, only 90% of available torque will be used to stop the horizontal motion. Finally, after the approach clearance position is achieved, the Z axis will be decelerated using 98% of available torque to come to rest at the final destination. To compute the motion speeds, the system will utilize the dynamic coefficients that were stored into model set number 1 by the SFUTIL utility program.

```
MOVEF pick, 10, 15, 99, 100, 90, 98, 1
```

> **NOTE:** The torque values 99 and 98 are used only to clarify the relationship between the example instruction and the text above. For an actual application, these torque values would likely be set to 100.

Related Keywords

APPRO and **APPROS**	(program instructions)
DEPART and **DEPARTS**	(program instructions)
MOVE and **MOVES**	(program instructions)
MOVET and **MOVEST**	(program instructions)
SELECT	(program instruction and real-valued function)

Syntax

```
MOVET location, hand_opening

MOVEST location, hand_opening
```

Function

Initiate a robot motion to the position and orientation described by the given location and simultaneously operate the hand.

Usage Considerations

These instructions are usually entered into programs with the V+ editor teach modes (see EDIT monitor command).

MOVET causes a joint-interpolated motion.

MOVEST causes a straight-line motion, during which no changes in configuration are permitted.

These instructions can be executed by any program task so long as the task has attached a robot. The instructions apply to the robot selected by the task.

If the V+ system is not configured to control a robot, executing these instructions will cause an error.

Parameters

`location` Transformation, precision point, location function, or compound transformation that specifies the destination to which the robot is to move.

`hand_opening` Real-valued expression that specifies whether the hand is to be closed (value zero) or opened (value nonzero).

Details

These instructions are similar to MOVE and MOVES, respectively.

One difference with these instructions is that the hand opening is controlled during the robot motion. The pneumatic-control system receives an open signal if hand opening is greater than zero. Otherwise, it receives a close signal. The hand opening is changed to hand opening millimeters during the motion if a servo-controlled hand is used.

Another difference with these instructions is that they can be entered into programs with the editor teach modes. While in T teach mode, every time the REC/DONE button on the manual control is pressed a MOVET instruction is entered in the program being edited. The specified location variable is set equal to the location of the robot at that instant and the hand opening is recorded similarly.

MOVEST instructions can be entered in the same way using the editor TS teach mode.

Examples

Move by joint-interpolated motion to the location described by the transformation part1, opening the hand during the motion:

```
MOVET part1, 1
```

Move along a straight-line path to the location described by the transformation part[7], closing the hand during the motion:

```
MOVEST part[7], 0
```

Related Keywords

APPRO and **APPROS** (program instructions)

DEPART and **DEPARTS** (program instructions)

MOVE and **MOVES** (program instructions)

MOVEF and **MOVESF** (program instructions)

SELECT (program instruction and real-valued function)

Syntax

```
MULTIPLE ALWAYS
```

Function

Allow full rotations of the robot wrist joints.

Usage Considerations

Only the next robot motion will be affected if the ALWAYS parameter is not specified.

This is the default state of the V+ system. MULTIPLE ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The MULTIPLE instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the MULTIPLE instruction will cause an error.

Parameter

ALWAYS	Optional qualifier that establishes MULTIPLE as the default condition. That is, if ALWAYS is specified, MULTIPLE will remain in effect continuously until disabled by a SINGLE instruction. If ALWAYS is not specified, the MULTIPLE instruction will apply only to the next robot motion.

Details

While MULTIPLE is in effect, full rotations of the wrist joints are used as required during motion planning and execution.

Related Keywords

CONFIG	(real-valued function)
SELECT	(program instruction and real-valued function)
SINGLE	(program instruction)

Syntax

NETWORK (component,code**)**

Function

Return network status and IP address information

Usage Considerations

This function is intended for Adept MV controllers fitted with the AdeptNet option. It does, however, return correct status information when the AdeptNet option is not present.

Parameters

component Real-valued expression that identifies the component of the AdeptNet system that is of interest.

1 = TCP
2 = NFS
3 = FTP

code Real-valued expression that further identifies the information desired. This value is ignored for NFS and FTP. For TCP:

0 = Return status value as described later (default).
1 = Return AD1*256 + AD2
2 = Return AD3*256 + AD4

where AD*n* is the nth byte of the IP address

Details

This function returns one of the following values if status is requested (that is, if the code argument is omitted or set to 0):

Value	Meaning
0	Hardware not present, or license not installed
–1	Hardware present and license detected
1	Driver is running

Syntax

NEXT count

Function

Branch to the END statement of the nth nested loop, perform the loop test, and loop if appropriate.

Usage Considerations

This instruction can be used with the FOR, WHILE, and DO control structures.

Parameter

count	Optional integer specifying the number of nested structures to branch to the END of (expressions and variables are not acceptable).

Details

When a NEXT instruction is processed with count = 1, execution continues at the END of the control structure. If count > 1, execution continues at the END of count number of nested control structures.

Example

If error = 1, branch to the END of the innermost control structure. If error = 2, branch to the END of the outermost control structure:

```
45 FOR i = 1 to 20
46   FOR j = 1 to 10
47     FOR k = 10 to 50
48       IF error == 1 THEN
49         NEXT ;branch to step 54
50       END
51       IF error == 2 THEN
52         NEXT 3 ;branch to step 56
53       END
54     END
55   END
56 END
57
```

Related Keywords

DO...UNTIL	(program instruction)
EXIT	(program instruction)
FOR	(program instruction)
WHILE...DO	(program instruction)

Syntax

NOFLIP

Function

Request a change in the robot configuration during the next motion so that the pitch angle of the robot wrist has a positive value.

Usage Considerations

Configuration changes cannot be made during straight-line motions.

If the selected robot does not support a no-flip configuration, this instruction is ignored by the robot.

The NOFLIP instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the NOFLIP instruction causes an error.

See **Figure 2-3 on page 224**.

Related Keywords

CONFIG	(real-valued function)
FLIP	(program instruction)
SELECT	(program instruction and real-valued function)

Syntax

NONULL ***ALWAYS***

Function

Instruct the V+ system not to wait for position errors to be nulled at the end of continuous-path motions.

Usage Considerations

Only the next robot motion will be affected if the ALWAYS parameter is not specified.

NULL ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The NONULL instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the NONULL instruction will cause an error.

Parameter

ALWAYS	Optional qualifier that establishes NONULL as the default condition. That is, when ALWAYS is included in a NONULL instruction, NONULL will remain in effect continuously until disabled by a NULL instruction. If ALWAYS is not specified, the NONULL instruction will apply only to the next robot motion.

Details

When NONULL is in effect and a BREAK in the robot motion occurs, V+ does not wait for the electronics to signal that all moving joints have reached their specified positions before it begins the next motion. That is, at the end of the allotted time, V+ assumes that the joints have all reached their final positions and starts commanding the next motion.

Like COARSE mode, this mode allows faster motion if high final-position accuracy is not required. However, since no position-error checking is done, large position errors can occur.

Related Keywords

COARSE	(program instruction)
CONFIG	(real-valued function)
FINE	(program instruction)
NULL	(program instruction)
SELECT	(program instruction and real-valued function)

Syntax

```
NOOVERLAP ALWAYS
```

Function

Generate a program error if a motion is planned that will cause selected multiturn axes to turn more than ± 180 degrees (the long way around) in order to avoid a limit stop.

Usage Considerations

NOOVERLAP affects only the operation of joints 1, 4, and 6 of the PUMA robot module and joint 4 of all SCARA robot modules.

Parameter

ALWAYS	Optional qualifier that establishes NOOVERLAP as the default condition. That is, if ALWAYS is specified, NOOVERLAP remains in effect continuously until disabled by an OVERLAP instruction. If ALWAYS is not specified, the NOOVERLAP instruction applies only to the next robot motion.

Details

When NOOVERLAP is set, and the transformation destination of the next joint-interpolated or straight-line motion requires that a multiple-turn axis rotate more than ± 180 degrees, the motion is not executed and a program error is generated. If the destination is specified as a precision point, this test is not performed.

In general, given a transformation destination, a multiple-turn axis normally attempts to move to a new position that is within ± 180 degrees of the previous motion's destination. The only conditions that force an axis to make a larger change is if SINGLE is specified (see following paragraph), or if a software limit stop would be violated.

When NOOVERLAP is set, the setting of SINGLE or MULTIPLE mode is ignored.

As with other user program errors, the error condition generated as a result of the NOOVERLAP test can be trapped by a standard REACTE subroutine if desired.

Related Keywords

MULTIPLE (program instruction)

OVERLAP (program instruction)

SINGLE (program instruction)

Syntax

```
NORMAL transformation_value)
```

Function

Correct a transformation for any mathematical round-off errors.

Usage Considerations

For most robot programs, transformation normalizing never has to be performed.

Parameter

`transformation_value` Transformation, transformation valued function, or compound transformation whose value is to be normalized.

Details

Use this function after a lengthy series of computations that modifies a transformation value. For instance, a procedural motion that incrementally changes the orientation of a transformation should occasionally normalize the resultant value. Within a transformation, the orientation of the robot is represented by three perpendicular unit vectors. Due to the small inaccuracies that occur in computer computations, after being incrementally modified many times, these vectors can become nonperpendicular or not of unit length.

The NORMAL function returns a transformation value that is essentially the same as the input argument but has the orientation portion of the value corrected for any small buildup of computational errors that may have occurred.

Syntax

```
... NOT value ...
```

Function

Perform logical negation of a value.

Usage Considerations

The word not cannot be used as a program name or variable name.

Details

The NOT operator operates on a single value, converting it from logically true to false, and vice versa. If the single value is zero, a –1.0 (TRUE) is returned. Otherwise, a 0.0 (FALSE) value is returned.

Refer to the *V+ Language User's Guide* for the order in which operators are evaluated within expressions.

Examples

```
IF NOT initialized THEN ;If the variable "initialized" has a
   CALL appl.setup()    ;FALSE value, the instructions in the
   initialized = TRUE   ;IF structure will be executed.
END
```

Syntax

```
... NOT.CALIBRATED
```

Function

Indicate (or assert) the calibration status of the robots connected to the system.

Usage Considerations

The current value of the NOT.CALIBRATED parameter can be determined with the PARAMETER monitor command or real-valued function.

You can modify the value of this parameter at any time; refer to the next section for details.

The parameter name can be abbreviated.

Details

The value of this parameter, which can range from –1 to 32767, should be interpreted as a bit mask. Bits 1 through 15 correspond to robots 1 through 15, respectively. For example, the following values have the following interpretations:

Value of parameter	Interpretation
0	All robots are calibrated.
1	Robot 1 is not calibrated.
3	Robots 1 and 2 are not calibrated.
7	Robots 1 through 3 are not calibrated.

On power-up, this parameter is set to indicate that all installed robots are not calibrated. If a robot is not connected or not defined, its NOT.CALIBRATED bit is always off.

The CALIBRATE command and instruction attempt to calibrate any enabled ROBOT that has its NOT.CALIBRATED bit set.

When the calibration operation completes, the NOT.CALIBRATED bits are updated as appropriate. For example, lets consider a system that has only one robot installed. If the CALIBRATE command is issued, and it succeeds, then NOT.CALIBRATED is set to 0. If three robots are connected, and the CALIBRATE command succeeds in calibrating robots 1 and 2, but not robot 3, then NOT.CALIBRATED is set to 4 (binary 100—robots 1 and 2 calibrated, 3 not calibrated).

The purpose of this parameter is to allow one of the bits to be *set* to force the corresponding robot to be calibrated the next time a CALIBRATE command or instruction is executed. This parameter can also be used to determine the calibration status of the robot(s).

The parameter value can be changed at any time. The following rules describe how a new asserted value is treated:

- If the new value asserts that a robot is not calibrated, the V+ system acts as if the robot is not calibrated whether or not the servo software believes that the robot is not calibrated.
- If the new value asserts that a robot is calibrated, the servo software is checked and V+ tracks the calibrated/not calibrated state indicated by the servo software for that robot. (Note that it is usually not meaningful to use PARAMETER NOT.CALIBRATED to clear a bit.)

Examples

Mark all installed robots as uncalibrated:

```
PARAMETER NOT.CALIBRATED = -1
```

Mark only robots 1 and 2 as uncalibrated:

```
PARAMETER NOT.CALIBRATED = 3
```

Related Keywords

CALIBRATE (monitor command and program instruction)

PARAMETER (monitor command, program instruction, and real-valued function)

ROBOT (system switch)

Syntax

NULL ***ALWAYS***

Function

Instruct the V+ system to wait for position errors to be nulled at the end of continuous path motions.

Usage Considerations

Only the next robot motion will be affected if the ALWAYS parameter is not specified.

This is the default state of the V+ system. NULL ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The NULL instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the NULL instruction will cause an error.

The word null cannot be used as a program name or variable name.

Parameter

ALWAYS	Optional qualifier that establishes NULL as the default condition. That is, if ALWAYS is specified, NULL will remain in effect continuously until disabled by a NONULL instruction. If ALWAYS is not specified, the NULL instruction will apply only to the next robot motion.

Details

When NULL is in effect and a BREAK in the robot motion occurs, V+ waits for the electronics to signal that all moving joints have reached their specified positions before it begins the next motion. The accuracy to which the electronics verify that all joints have reached their destination positions is determined by the COARSE and FINE program instructions.

Related Keywords

COARSE	(program instruction)
FINE	(program instruction)
NONULL	(program instruction)
NULL	(transformation function)
SELECT	(program instruction and real-valued function)

Syntax

NULL

Function

Return a null transformation value—one with all zero components.

Usage Considerations

The word null cannot be used as a program name or variable name.

Details

A null transformation corresponds to a null vector (X = Y = Z = 0) and no rotation (yaw = pitch = roll = 0). Such a transformation is useful, for example, with a SHIFT function to create a transformation representing a translation with no rotation.

Example

Define a new transformation (new.loc) to be the result of shifting an existing transformation (old.loc) in the World coordinate directions.

```
new.loc = SHIFT(NULL BY x.shift,y.shift,z.shift):old.loc
```

Determine the length of the vector described by the transformation test.loc.

```
dist = DISTANCE(NULL, test.loc)
```

Related Keywords

CONFIG (real-valued function)

NULL (program instruction)

Syntax

OFF

Function

Return the value used by V+ to represent a logical false result.

Usage Considerations

The word off cannot be used as a program name or variable name.

Details

This named constant is useful for situations where on and off conditions need to be specified. The value returned is 0.

This function is equivalent to the FALSE function.

Related Keywords

FALSE (real-valued function)

ON (real-valued function)

Syntax

ON

Function

Return the value used by V+ to represent a logical true result.

Usage Considerations

The word on cannot be used as a program name or variable name.

Details

This named constant is useful for situations where on and off conditions need to be specified. The value returned is –1.

This function is equivalent to the TRUE function.

Related Keywords

OFF (real-valued function)

TRUE (real-valued function)

Syntax

OPEN

OPENI

Function

Open the robot gripper.

Usage Considerations

OPEN causes the hand to open *during* the next robot motion.

OPENI causes a BREAK in the current continuous-path motion and causes the hand to open *immediately* after the current motion completes.

The OPEN instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

The OPENI instruction can be executed by any program task so long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing these instructions will cause an error.

Details

These instructions cause the control valves for the pneumatic hand to receive a signal to open. If the OPEN instruction is used, the signal will be sent when the next robot motion begins.[1]

The OPENI instruction differs from OPEN in the following ways:

- A BREAK occurs if a continuous-path robot motion is in progress.
- The signal is sent to the control valves at the conclusion of the current motion or immediately if no motion is in progress.
- Robot motions are delayed for a brief time to allow the hand actuation to complete. The length of the delay (in seconds) is the current setting of the HAND.TIME system parameter.

[1] You can use the Robot Configuration Utility program to set the digital signals that control the pneumatic hand. The utility program is in the file CONFIG_R.V2 on the Adept Utility Disk. See the manual ***Instructions for Adept Utility Programs*** for information on use of the program.

Examples

During the next robot motion, cause the pneumatic control valves to assume the open state:

```
OPEN
```

Cause the pneumatic control values to assume the open state at the conclusion of the current motion:

```
OPENI
```

Related Keywords

CLOSE and **CLOSEI**	(program instructions)
HAND.TIME	(system parameter)
RELAX and **RELAXI**	(program instructions)
SELECT	(program instruction and real-valued function)

Syntax

```
... value OR value ...
```

Function

Perform the logical OR operation on two values.

Details

The OR operator operates on two values, resulting in their logical OR combination. For example, during the OR operation

```
c = a OR b
```

the following four situations can occur:

a	b		c
FALSE	FALSE	➡	FALSE
FALSE	TRUE	➡	TRUE
TRUE	FALSE	➡	TRUE
TRUE	TRUE	➡	TRUE

That is, the result is TRUE if either (or both) of the two operand values is logically TRUE.

Refer to the *V+ Language User's Guide* for the order in which operators are evaluated within expressions.

Example

In the following sequence, the instructions immediately following the IF instruction are executed if either ready is TRUE (that is, nonzero) or count equals 1. The instructions are not executed if both ready is FALSE and count is not equal to 1.

```
IF ready OR (count == 1) THEN
   .
   .
   .
END
```

Related Keywords

AND (operator)

BOR (operator)

XOR (operator)

Syntax

```
OUTSIDE (low, test, high)
```

Function

Test a value to see if it is outside a specified range.

Parameters

low	Real value, expression, or variable specifying the lower limit of the range to be tested.
test	Real value, expression, or variable to test against the range.
high	Real value, expression, or variable specifying the upper limit of the range to be tested.

Details

Returns TRUE (–1) if **test** is less than **low** or greater than **high**. Returns FALSE (0) otherwise.

Related Keywords

MAX	(real-valued function)
MIN	(real-valued function)

Syntax

OVERLAP ***ALWAYS***

Function

Disable the NOOVERLAP limit-error checking either for the next motion or for all subsequent motions.

Usage Considerations

OVERLAP applies only to the operation of joints 1, 4, and 6 of the PUMA robot module and joint 4 of all SCARA robot modules.

Parameter

ALWAYS	Optional qualifier that establishes OVERLAP as the default condition. That is, if ALWAYS is specified, OVERLAP remains in effect continuously until disabled by a NOOVERLAP instruction. If ALWAYS is not specified, the OVERLAP instruction applies only to the next robot motion.

Details

When OVERLAP is set, and the transformation destination of the next joint-interpolated or straight-line motion requires that a multiple-turn axis rotate more than ± 180 degrees, the motion is executed without generating a program error message.

OVERLAP disables the limit-error checking of NOOVERLAP. The overlap condition is set when the V^+ system is loaded from disk.

Related Keyword

NOOVERLAP (program instruction)

Syntax

```
PACK string_array[index], first_char, num_chars = string

PACK string_var, first_char, num_chars = string
```

Function

Replace a substring within an array of (128-character) string variables, or within a (nonarray) string variable.

Parameters

`string_array` String array variable that is modified by the substring on the right-hand side of the equal sign. Each element within the string array is assumed to be 128 characters long (see below).

`index` Optional integer value that identifies the first array element to be considered. The **`first_char`** value is interpreted relative to the element specified by this index. If no index is specified, element zero is assumed.

`string_var` String variable that is modified by the substring on the right-hand side of the equal sign.

`first_char` Real-valued expression that specifies the position of the first character of the substring within the string array. A value of 1 corresponds to the first character of the specified string array element. This value must be greater than zero.

The value of **`first_char`** can be greater than 128. In that case the array element accessed will follow the element specified in the function call. For example, a value of 130 corresponds to the second character in the array element following that specified by `index`.

`num_chars` Real-valued expression that specifies the number of characters to be copied from the string to the array. This value can range from 0 to 128.

`string` String variable, constant, or expression from which the substring is to be extracted. The string must be at least **`num_chars`** long.

Details

This instruction replaces a substring within an array of strings or within a string variable. When an array of strings is being modified, the substring is permitted to overlap two elements of the string array. For example, a 10-character substring whose first character is to replace the 127th character in element [3] will supersede the last two characters in element [3] and the first eight characters of element [4].

If the array element to be modified is not defined, the element will be created and filled with ASCII NUL characters (^H00) up to the specified start of the substring. Similarly, if the array element to be modified is too short, the string will be padded with ASCII NUL characters to the start of the substring.

In order to efficiently access the string array, this function assumes that all of the array elements, from the start of the array until the element before the element accessed, are defined and are 128 characters long. For multidimensional arrays, only the right-most array index is incremented to locate the substring. Thus, for example, element [2,3] is followed by element [2,4].

When a string variable is modified, the replacement is done in a manner similar to that for an individual element. However, in this case, there are two differences to the operation of the instruction:

- If the initial value of the string variable is shorter than **`first_char`** characters, the string is not extended with ASCII NUL characters. That is, the replacement string is simply appended to the value of the variable.
- An error results if the operation would cause the string to be longer than 128 characters.

Example

The instruction below replaces 11 characters within the string array $list[ ]. The replacement is specified as starting in array element $list[3]. However, since the first character replaced is to be number 130, the 11-character substring will actually replace the second through 12th characters of $list[4].

```
PACK $list[3], 130, 11 = $string
```

Related Keywords

$MID (string function)

$UNPACK (string function)

Syntax

PARAMETER parameter_name = value

PARAMETER parameter_name[index**] = value**

Function

Set the value of a system parameter.

Usage Considerations

If the specified parameter accepts an index qualifier and the index is zero or omitted (with or without the brackets), *all* the elements of the parameter array are assigned the value given.

Parameters

parameter_name Name of the parameter whose value is to be modified.

index	For parameters that can be qualified by an index, this is an optional real value, variable, or expression that specifies the specific parameter element of interest (see above).
value	Real value, variable, or expression defining the value to be assigned to the system parameter.

Details

This instruction sets the given system parameter to the value on the right. The parameter name can be abbreviated to the minimum length that identifies it uniquely.

> **NOTE:** A regular assignment statement cannot be used to set the value of a system parameter.

The parameter names acceptable with the standard V+ system are summarized in the ***V+ Language User's Guide***. The Parameter entry in the index for this document directs you to the detailed descriptions of these parameters.

Other system parameters are available when options are installed. Refer to the option documentation for details. For example, the parameters associated with the AdeptVision options are described in the *AdeptVision Reference Guide*.

Example

```
;Set the TERMINAL system parameter to 4.
PARAMETER TERMINAL = 4
```

Related Keywords

BELT.MODE	(system parameter)
HAND.TIME	(system parameter)
KERMIT.RETRY	(system parameter)
KERMIT.TIMEOUT	(system parameter)
NOT.CALIBRATED	(system parameter)
PARAMETER	(monitor command and program instruction)
SCREEN.TIMEOUT	(system parameter)
TERMINAL	(system parameter)

Syntax

PARAMETER (parameter_name)

PARAMETER (parameter_name[index])

Function

Return the current setting of the named system parameter.

Parameters

parameter_name Name of the system parameter whose value is to be returned.

index For parameters that can be qualified by an index, this is a (required) real value, variable, or expression that specifies the specific parameter element of interest.

Details

This function returns the current setting of the given system parameter. The parameter name can be abbreviated to the minimum length that identifies it uniquely.

Other system parameters are available when options are installed. Refer to the option documentation for details. For example, the parameters associated with the AdeptVision options are described in the *AdeptVision Reference Guide*.

Examples

The following example illustrates how the current setting of the TERMINAL parameter can be displayed on the system terminal during program execution:

```
TYPE "The TERMINAL parameter is set to", PARAMETER(TERMINAL)
```

The PARAMETER function can also be used in any expression to include the value of a parameter. For example, the following program statement could be used to increase the time delay for hand actuation:

```
PARAMETER HAND.TIME = PARAMETER(HAND.TIME) + 0.15
```

Note that the left-hand occurrence of PARAMETER is the instruction name and the right-hand occurrence is the function name.

Related Keywords

BELT.MODE	(system parameter)
HAND.TIME	(system parameter)
KERMIT.RETRY	(system parameter)
KERMIT.TIMEOUT	(system parameter)
NOT.CALIBRATED	(system parameter)
PARAMETER	(monitor command and program instruction)
SCREEN.TIMEOUT	(system parameter)
TERMINAL	(system parameter)

Syntax

PAUSE

Function

Stop program execution but allow the program to be resumed.

Usage Considerations

Unlike HALT and STOP, the PAUSE instruction does not force FCLOSE or DETACH on the disk or serial communication logical units. If the program has a file open and you decide not to continue execution of the current program, you should issue a KILL command (with the appropriate task number) to close all files and detach all logical units.

Details

Causes a BREAK and then terminates execution of the application program, displaying the message (PAUSED). Execution can subsequently be continued by typing **proceed** and the appropriate task number, and pressing the RETURN key.

When debugging a program, a PAUSE instruction can be inserted to stop program execution temporarily while the values of variables are checked.

> **NOTE:** Any robot motion in progress when a PAUSE instruction is processed will complete normally.

Related Keywords

HALT	(program instruction)
KILL	(monitor command and program instruction)
PROCEED	(monitor command, see the *V+Operating System Reference Guide)*
STOP	(program instruction)

Syntax

```
PAYLOAD value, motor
```

Function

Adjust the feedforward compensation for a specified motor by setting a percentage of the maximum payload assumed for that motor.

Usage Considerations

This instruction affects the robot currently selected with a SELECT program instruction.

A task does not have to be attached to the robot to issue this instruction.

With Adept robots, this instruction is supported only on motors 3 and 4 of the AdeptOne-MV, the AdeptThree-MV, and the PackOne-MV robots. The instruction is useful primarily for tuning motor 4 performance as a function of end-effector inertia. With AdeptMotion VME robots, the instruction defaults to no effect and must be enabled with the SPEC utility.

Parameters

`value`	An optional real-valued expression interpreted as a percentage of the maximum payload. If omitted, the value –1 is used, which is interpreted as the default payload percentage.
`motor`	An optional real-valued expression that specifies to which motor this instruction applies. If the parameter is omitted or zero, the payload percentage applies to all motors of the selected robot.

Details

The PAYLOAD instruction takes effect immediately and is not synchronized with robot-motion segments.

If the PAYLOAD instruction is used to indicate that a very large payload is on the robot motor when in fact a small payload is on the robot motor, the motor can run away any time the commanded acceleration is nonzero. The degree of run-away depends upon the range between minimum and maximum acceleration feedforward, the degree to which the payload is in error, and the size of the acceleration and deceleration during the motion being attempted. In particular, this is a concern for motor 4 of the AdeptOne-MV, the AdeptThree-MV, and the PackOne-MV robots. Take care to ensure that the value used with the PAYLOAD instruction correctly reflects the true payload.

On the AdeptOne-MV, the AdeptThree-MV, the PackOne-MV, and the Adept 550 robots:

- Only motor 3 drives joint 3.
- Motor 4 drives joint 4 with a small contribution from motor 3.

You can use the PAYLOAD instruction to adjust feedforward gains in a predetermined way, just as you can use the GAIN.SET instruction to adjust feedback gains in a predetermined way. Because GAIN.SET does not affect the feedforward, there is no conflict between these two instructions. That is, the order of their execution is unimportant.

The `value` parameter specifies a value of the acceleration feedforward gain (inertia estimate) that is interpolated between two factory preset values. These are the minimum no-load acceleration feedforward gain and the maximum acceleration feedforward gain.

On AdeptMotion VME robots, this instruction defaults to no effect. Using the SPEC utility program, you can enable it for a particular AdeptMotion VME motor by setting the maximum and minimum acceleration feedforward values for that motor to nonzero values (with the maximum greater than the minimum). The PAYLOAD instruction assumes the use of current-mode (torque-mode) amplifiers.

The benefit of the instruction depends upon the mechanical design of the mechanism. Motors with low gear ratios and high variations in payload benefit more. Motors with high gear ratios and low variations in payload benefit less. The instruction is not useful with velocity-mode amplifiers.

Related Keywords

ACCEL	(program instruction and real-valued function)
DURATION	(program instruction and real-valued function)
GAIN.SET	(program instruction)
SPEED	(monitor command, program instruction, and real-valued function)

Syntax

#PDEST

Function

Return a precision-point value representing the planned destination location for the current robot motion.

Usage Considerations

The #PDEST function returns information for the robot selected by the task executing the function.

If the V+ system is not configured to control a robot, use of the #PDEST function will not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

The name pdest cannot be used for a variable or a program.

Details

The #PDEST function can be used to determine where the robot was moving to after its motion is interrupted for some reason.

The #PDEST function is equivalent to the DEST transformation function and can be used interchangeably with DEST, depending upon the type of location information that is desired. Please refer to the description of the DEST function for more information on the use of both the #PDEST and DEST functions.

Related Keywords

DEST (transformation function)

HERE (transformation function)

SELECT (program instruction and real-valued function)

Syntax

PENDANT (select)

Function

Return input from the manual control pendant.

Usage Considerations

The manual control pendant logical unit must be attached, and the pendant must be in USER mode, before keys can be read or output to the pendant can be done. Thus, the application program should use the PENDANT function to check these conditions before performing pendant I/O.

Parameter

select Real-valued expression whose value selects what type of pendant information is returned (see below).

Details

The value returned depends upon the select parameter as follows:

select > 0 Immediately returns a value that reflects the actual state of the key with the given key number at the instant the function is called. The state of the key depends upon the key mode setting for that key. See the KEYMODE program instruction for information about setting key modes and for a table of the key numbers. The value returned is meaningful only if the pendant is connected. (The pendant logical unit does not need to be attached for this mode of operation.)

If a key is in keyboard mode, the value ON (–1) indicates that the key is pressed. The value OFF (0) indicates that the key is not pressed.

If a key is in level mode, the value ON (–1) indicates that the pendant is attached in USER mode and that the key is pressed. The value OFF (0) indicates that the pendant is not in USER mode, or that the key is not pressed.

If a key is in toggle mode, the value ON (–1) indicates that the key is on and the value OFF (0) indicates that the key is off. If the pendant is not in USER mode, the value returned still accurately reflects the state of the toggled key.

NOTE: When **select** is equal to 36 (indicating the SLOW key), the value returned indicates the current state of slow mode, as indicated by the LED on the key. (The SLOW key is always in toggle mode and is not affected by the KEYMODE command.)

select = 0 Returns the key number of the next keyboard mode key pressed. Program execution is suspended until a keyboard mode key is pressed. If no key is programmed in this mode, an error occurs. The pendant logical unit must be attached for this mode of operation. (See the ATTACH program instruction.)

select = –1 Returns the key number of the next special mode key pressed. Program execution is suspended until a key of the requested mode is pressed. If no key is programmed in this mode, an error occurs. The pendant logical unit must be attached for this mode of operation. (See the ATTACH program instruction.)

select = –2 Returns the current value from the speed potentiometer, in the range of –128 (decimal) to 127 (decimal). (The pendant logical unit does not need to be attached for this mode of operation.)

select = –3 Returns the current display mode active on the manual control. This can be used, for example, to determine the state of the manual control before attempting to write to it. (The pendant logical unit does not need to be attached for this mode of operation.)

The display modes should be interpreted as follows:

Display mode	Interpretation
1	Function display (e.g., DISP)
2	Background display
3	Error display
4	USER mode

select = –4 Returns the version number of the manual control software. This is the same is the value returned by the real-valued function ID(1,2). The value –1 is returned if the manual control pendant is not connected to the system.

Examples

This example sets the manual control soft keys to keyboard mode, and then waits for one of them to be pressed (also see the *V+ Language User's Guide*).

```
ATTACH (1)              ;Attach the pendant LUN
KEYMODE 1,5 = 0         ;Set soft keys to keyboard mode
key = PENDANT(0)        ;Wait and return next key hit
TYPE "Soft key #", key, " pressed"
DETACH (1)              ;Detach the pendant LUN
```

This example sets the DONE key to level mode and loops until the key is pressed.

```
ATTACH (1)              ;Attach the pendant LUN
KEYMODE 8 = 2           ;Set DONE key (8) to level mode
WAIT PENDANT(8)         ;Pause until DONE key is pressed
DETACH (1)              ;Detach the pendant LUN
```

Related Keywords

ATTACH (program instruction)

KEYMODE (program instruction)

Syntax

PI

Function

Return the value of the mathematical constant pi (3.141593).

> **NOTE:** TYPE, PROMPT, and similar instructions display the result of the above example as a single-precision value. However, pi is actually stored and manipulated as a double-precision value. The LISTR monitor command displays real values to full precision.

Usage Considerations

The word pi cannot be used as a program name or variable name.

Syntax

```
#PLATCH (select)
```

Function

Return a precision-point value representing the location of the robot at the occurrence of the last external trigger or AdeptForce guarded-mode trigger.

Usage Considerations

The function name #PLATCH is considered to be a precision-point name. Thus, the # character must precede all uses of the function.

#PLATCH( ) returns information for the robot selected by the task executing the function. If the V+ system is not configured to control a robot, use of the #PLATCH function will not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

Parameter

`select` Optional integer, expression, or real variable specifying:

0 External trigger (default)
1 AdeptForce guarded mode-trigger

Details

#PLATCH( ) returns a precision-point value that represents the location of the robot when the last trigger occurred. The LATCHED real-valued function should be used to determine when an external trigger has occurred and a valid location has been recorded.

Operation of the external trigger can be configured with the Adept controller configuration program (in the file CONFIG_C.V2 on the Utility Disk). This trigger may originate from the vision processor or a digital input signal.

See the ***AdeptForce VME User's Guide*** for details of the AdeptForce option.

Related Keywords

LATCH (transformation function)

LATCHED (real-valued function)

Syntax

```
POS (search_string, sub_string, start)
```

Function

Return the starting character position of a substring in a string.

Parameters

`search_string` String expression to be searched for the occurrence of a substring.

`sub_string` String expression containing the substring to be searched for within the search string.

`start` Optional expression indicating the character position within the search string where searching is to begin.

Details

Returns the character position in **`search_string`** where **`sub_string`** begins. If the substring does not occur within the search string, a value of 0 is returned.

If `start` is provided, it indicates the character position within **`search_string`** where searching will begin. A value of 1 indicates the first character. If `start` is omitted or less than 1, searching begins with the first character. If start is greater than the length of **`search_string`**, a value of 0 is returned.

When checking for a matching substring, uppercase and lowercase letters are considered to be the same.

Examples

```
POS("file.ext", ".")    ;Returns 5
POS("file", ".")        ;Returns 0
POS("abcdefgh", "DE")   ;Returns 4
POS("1-2-3-4", "-", 5)  ;Returns 6
```

Syntax

```
... POWER
```

Function

Control or monitor the status of high power.

Usage Considerations

From the system terminal or with a program, the POWER switch can be used to turn off high power or to start the process to turn it on.

Only CPU #1 can enable and disable power.

WARNING: For systems operating under V+ versions prior to 11.3 and not subject to European certification, Adept recommends that this switch not be used to turn on high power from within a program.

Using this switch to turn on high power is potentially dangerous when performed from a program because the robot can be activated without direct operator action. Turning on high power from the terminal can be hazardous if the operator does not have a clear view of the robot workspace or does not have immediate access to an Emergency Stop button.

For systems operating under V+ 11.3 with or without the Manual Mode Safety Package (MMSP), turning on high power from the terminal can be hazardous if the operator does not have a clear view of the robot workspace or does not have immediate access to an Emergency Stop button.

Details

Enabling this switch is equivalent to pushing the COMP/PWR button on the manual control pendant to turn on high power. If there is no error condition that prevents power from coming on, the enabling process proceeds to the second step, in which you must press the HIGH POWER ON/OFF button on the VFP. (Systems operating under V+ versions prior to 11.3 do not require the second step.)

Disabling this switch requests the robot to perform a controlled deceleration and power-down sequence. This sequence consists of:

1. Decelerating all robots according to the user-specified parameters.[1]
2. Turning on the brakes.
3. Waiting for the user-specified brake-delay interval.[1]
4. Turning off the amplifiers and power.
5. Asserting the backplane Emergency Stop signal and deasserting the High Power Enable (HPE) signal.

Note that DISABLE POWER may take an arbitrarily long time due to long deceleration times and long brake turn-on delays. (Use the ESTOP command or program instruction when you desire an immediate shutdown.) The value of this switch can be checked at any time with the SWITCH real-valued function to determine if high power is on or off.

To disable power from a robot program without generating an error condition, the program either must be in DRY.RUN mode or it must DETACH the robot from program control. See the DRY.RUN switch or DETACH program instruction for details.

Example

The following program segment detaches the robot, turns high power off, and waits for the operator to turn high power back on.

```
DETACH              ;Detach robot from program
DISABLE POWER       ;Turn off power
TYPE "Press the COMP/PWR button to continue"
ATTACH              ;Wait for power on and attach
TYPE "Robot program continuing..."
```

Related Keywords

DISABLE (monitor command and program instruction)

ENABLE (monitor command and program instruction)

ESTOP (program instruction)

SWITCH (monitor command, program instruction, and real-valued function)

[1] You set user-specified parameters with the SPEC utility program. Robot manufacturers set these parameters for some robots; if so, you cannot change them.

Syntax

```
#PPOINT (j1_value, j2_value, j3_value, j4_value, j5_value,
j6_value)
```

Function

Return a precision-point value composed from the given components.

Usage Considerations

The #PPOINT function name is considered to be a precision-point name. Thus, the # character must precede all uses of the function.

Parameters

`j1_value`
`j2_value`
`j3_value`
`...`
`j6_value`
	Optional real-valued expressions for the respective robot joint positions. (If more values are specified than the number of robot joints, the extra values are ignored.)

Details

Returns a precision-point value composed from the given components, which are the positions of the first through last robot joints, respectively.

A zero value is assumed for any parameter that is omitted.

Examples

Assume that you want to perform a coordinated motion of joints 2 and 3 of a robot with 4 joints, starting from its current location. The following program segment will perform such a motion:

```
HERE #ref               ;Define current location
DECOMPOSE x[] = #ref    ;Fill array with components

;Move to new precision point defined with modified
;components

MOVE #PPOINT(x[0], x[1]+a, x[2]-a/2, x[3])
```

The following steps would lead to the same final location, but the robot joints would not be moved simultaneously with this method.

```
DRIVE 2, a, 100     ;Drive joint 2
DRIVE 3, -a/2, 100  ;Drive joint 3
```

Related Keyword

TRANS (transformation function)

Syntax

PRIORITY

Function

Return the current reaction lock-out priority for the program.

Usage Considerations

The name priority cannot be used as a program name or variable name.

This function returns the reaction lock-out priority, not the program priority of the executing program.

Details

The reaction lock-out priority for each program task is set to zero when execution of the task is initiated. The priority can be changed by the program at any time with the LOCK instruction, or the priority is set automatically when a reaction occurs as prescribed by a REACT or REACTI instruction.

The PRIORITY function can be used to determine the current setting of the reaction lock-out priority for the task executing the function.

Example

This example raises the priority, performs some operation that requires a reaction routine to be locked out, and then restores it to its previous value.

```
save = PRIORITY     ;Save the current priority
IF save < 10 THEN   ;Raise priority to at least 10
   LOCK 10
END
; Access data shared by a reaction routine.
LOCK save           ;Set priority to original value
```

Related Keywords

LOCK (program instruction)

REACT (program instruction)

REACTI (program instruction)

Syntax

```
.PROGRAM program_name(argument_list) ;comment
```

Function

Define the arguments a program will be passed when it is invoked.

Usage Considerations

This instruction is inserted automatically by the V+ editors when a new program is edited.

This special instruction must be the first line of every program.

The .PROGRAM statement cannot be deleted from a program.

Parameters

`program_name` Name of the program in which this instruction is found.

`argument_list` Optional list of variable names, separated by commas. Each variable can be any one of the data types available with V+ (belt, precision point, real-value, string, and transformation). Each variable can be a simple variable or an array with all of its indexes left blank.

`;comment` Optional comment that will be displayed when the program is loaded from a disk file and when the DIRECTORY command is processed. (The semicolon [;] should be omitted if no comment is included.)

Details

The V+ editors automatically enter a .PROGRAM line when you edit a new program. They will also prevent you from deleting the line or changing the program name. You can, however, edit the line to add, delete, or modify the argument list. (The RENAME monitor command must be used to change the program name.)

The variables in the argument list will be considered automatic variables for the named program. (See the AUTO instruction.)

When a program begins execution (for example, via an EXECUTE command or instruction or a CALL instruction), the arguments in the .PROGRAM instruction are associated with those in the EXECUTE or CALL. This association allows values to be passed between a program and its caller.

See the description of the CALL instruction for an explanation of how the program arguments receive their values from a calling program and return their values to the calling program. The following rules apply to any program argument that is undefined when the program executes:

- Real-valued scalar parameters *can* be assigned a value within a program if they are undefined.
- Location, string, and belt (scalar or array) parameters, and real-valued array parameters, *cannot* be assigned a value within a program if they are undefined. (AUTO variables can be used to work around this restriction, as shown in the example below.)

  NOTE: If a program attempts to assign a value to one of these undefined variables, the error *Undefined value* will result. In that case, the error refers to the variable on the *left* side of the assignment instruction.

- If an undefined parameter is passed on to another program in a CALL instruction, and the type (real-valued or location) of the variable is ambiguous, the parameter will be assumed to be real-valued.
- Elements of an undefined array parameter cannot be passed by reference in a CALL instruction.

The DEFINED real-valued function can be used within a program to check whether or not a program parameter is defined. The example below shows how a program can be written to accommodate undefined parameters.

A comment can be included on the .PROGRAM line, which will be displayed when the program is loaded from the disk and by the DIRECTORY command.

Examples

Define a program that expects no arguments to be passed to it:

```
.PROGRAM get()
```

Define a program that expects a string-valued argument and either a location or real-valued argument (the type of the second argument will be determined by its use in the program):

```
.PROGRAM test($n, dx)
```

The following program segment shows how a program can be written to deal with undefined parameters. The example shows part of the program example, which has a real-valued parameter and a string parameter.

```
.PROGRAM example(real, $string)
  AUTO $internal.var

  ; Check for undefined real-valued scalar parameter.

  IF NOT DEFINED(real) THEN  ;If parameter is undefined
    real = 1                 ;assignment of desired
  END                        ;default value is okay

  ; Check for undefined string parameter.

  IF DEFINED($string) THEN   ;If parameter is defined,
    $internal.var = $string  ;use the parameter value
  ELSE      ;Otherwise,
    $internal.var = "default" ;use default value
  END

  ; (Program continues...)
```

Refer to the **DEFINED** function for more details and for testing nonreal arguments.

Related Keywords

CALL and **CALLS** (program instructions)

EXECUTE (monitor command and program instruction)

PRIME (monitor command)

SSTEP and XSTEP (monitor commands)

See the *V+ Operating System Reference Guide* for details on monitor commands.

Syntax

PROMPT output_string, **variable_list**

Function

Display a string on the system terminal and wait for operator input.

Parameters

output_string	Optional string expression that is output to the system terminal. The cursor is left at the end of the string.
variable_list	A list of real-valued variables, or a single string variable, that will receive the data.

Details

Displays the text of the output string on the system terminal, and waits for the user to type in a line terminated by pressing the RETURN key.

The input line can be processed in either of two ways:

1. If a list of real-valued variables is specified as the variable list, the line is assumed to contain a list of numbers separated by space characters and/or commas. Each number is converted from text to its internal representation, and its value is stored in the next variable contained in the variable list. If more values are read than the number of variables specified, the extra values are ignored. If fewer values are read, the remaining variables are set to zero. If data is read that is not a number, an error occurs and program execution stops. It is recommended that only one value be requested by each PROMPT instruction to avoid confusion and to reduce the possibility of error.
2. If a single string variable is specified as the variable list, the entire input line is stored in the string variable. The program must then process the string as appropriate.

If the user just presses the RETURN key, or presses CTRL+C, an empty line is read. That results in all the real variables being set to zero, or the string variable being assigned an empty string.

If the user presses CTRL+Z, an end-of-file error condition results. If there is no REACTE instruction active, program execution is terminated and an error message is displayed. Thus, CTRL+Z can be a useful way to abort program execution at a PROMPT.

Examples

Consider the instruction:

```
PROMPT "Enter the number of parts: ", part.count
```

The result of executing this instruction will be display of the message

```
Enter the number of parts:
```

on the system terminal to ask the operator to type in the desired value. After the user types a number and presses the RETURN key, the variable part.count will be set equal to the value typed, and program execution will resume.

Consider changing the above instruction to:

```
PROMPT "Enter the number of parts: ", $input
```

Even if the user enters characters that are not valid for numeric input, V+ will not output an error message. The application program can use the various string functions to extract numeric values from the input string.

If you want to include format specifications in the string output to the terminal (such as /Cn to skip lines), you can use either the $ENCODE function or the TYPE instruction. For example, the instruction

```
PROMPT $ENCODE(/B,/C1,/X10)+"Enter the number of parts: ",
$input
```

will beep the terminal, space down a line, space over ten spaces, output the string, and wait for the user's input. (Note that a + sign has to be used between the $ENCODE function and the quoted string because the entire **`output_string`** parameter must be a single string expression.)

The following pairs of instructions are equivalent to the previous example:

```
TYPE /B, /C1, /X10, /S
PROMPT "Enter the number of parts: ", $input
```

or

```
TYPE /B, /C1, /X10, "Enter the number of parts: ", /S
PROMPT , $input
```

Note that /S must be included in the TYPE instructions as shown to have the prompt string output on one line, and to have the cursor remain on that line.

Related Keywords

READ (program instruction)

TYPE (program instruction)

Syntax

RANDOM

Function

Return a pseudorandom number.

Usage Considerations

The word random cannot be used as a program name or variable name.

Details

Returns a pseudorandom number in the range 0.0 to 1.0, inclusive. Thus, each time the RANDOM function is evaluated, it returns a different value.

The numbers generated by this function are pseudorandom because the sequence will repeat after this function has been called 2^{24} (16,777,216) times.

Syntax

```
REACT signal_num, program, priority
```

Function

Initiate continuous monitoring of a specified digital signal and automatically trigger a subroutine call if the signal properly transitions.

Usage Considerations

The REACT (and REACTI) instruction can be executed by any of the program tasks. That is, each task can have its own, independent reaction definition.

Any of the first twelve external input signals (1001 to 1012) can be simultaneously monitored.

Reactions are triggered by signal *transitions* and not levels. Thus, if a signal is going to be monitored for a transition from off to on and the signal is already on when a REACT (or REACTI) instruction is executed, then the reaction will not occur until the signal goes off and then on again.

A signal must remain stable for at least 18 milliseconds to assure detection of a transition.[1]

The requested signal monitoring will be enabled as soon as a REACT (or REACTI) instruction is executed. Because of the way V+ processes program instructions, this could result in an effect on the motion initiated by the motion instruction *preceding* the REACT (or REACTI) instruction in the program. (See the ***V+ Language User's Guide*** for a discussion of robot motion processing.)

Parameters

`signal_num` Real-valued expression representing the signal to be monitored. The signal number must be in the range 1001 to 1012 (external input signals) or 2001 to 2008 (internal software signals). (The software signals can thus be used by one program task to interrupt another task.) If the signal number is positive, V+ looks for a transition from off to on; if **`signal_num`** is negative, V+ looks for a transition from on to off.

`program` Name of the subroutine that is to be called when the signal transitions properly.

[1] If software signals are being used to trigger reactions, the WAIT instruction (with no argument) should be used as required to ensure that the signal state remains constant for the required time period.

`priority`	Optional real-valued expression that indicates the relative importance of this reaction as explained below. The value of this expression is interpreted as an integer value and can range from 1 to 127. See the LOCK instruction for additional details on priority values. The default value is 1.

Details

When the specified signal transition is detected, V+ reacts by checking the priority specified with the REACT instruction against the program priority setting at that time. (The program priority is always set to 0 when execution begins. It can be changed with the LOCK instruction.) If the REACT priority is greater than the program priority, the normal program execution sequence is interrupted and the equivalent of a CALL program instruction is executed. Also, the program priority is temporarily raised to the REACT priority, locking out any reactions of equal or lower importance. When a RETURN instruction is executed in a reaction subroutine, the program priority is restored to the value it had before the reaction program was invoked.

If the REACT priority is less than or equal to the program priority when the signal transition is detected, the reaction is queued and will not occur until the program priority is lowered. Therefore, depending upon the relative priorities, there can be a considerable delay between the time a signal transition is noticed by V+ and the time the reaction program is actually invoked.

If multiple reactions are pending because of a priority lockout, the reaction with the highest priority will be serviced first when the locking priority is lowered. If multiple pending reactions have the same priority, the one associated with the highest signal number will be processed first.

The subroutine call to program is performed such that when a RETURN instruction is encountered, the next instruction to be executed will be the one that follows the last instruction processed before the reaction program was initiated. If there is a sequence of instructions that you do not want interrupted by a reaction program, you should use the LOCK instruction to raise the program priority during that sequence.

The signal monitoring continues until one of the following occurs:

- An IGNORE instruction is executed for the signal.
- A reaction occurs (in which case IGNORE signal_number is automatically performed).
- A REACT (or REACTI) instruction is executed that refers to the same signal. That is, if the signal specified in a REACT instruction is already being monitored by a previous REACT or REACTI instruction, the old instruction will be canceled when the new REACT instruction is executed.

Example

The instruction below monitors the external input signal identified by the value of the variable test. If the desired signal transition occurs (as specified by the sign of the value of test), program execution will branch to program delay as soon as the program priority drops to 0 (since no priority is specified in the instruction). (The program priority will be raised to 1 [the default value] when the subroutine is invoked; the program priority returns to 0 when the program returns.)

```
REACT test, delay
```

Related Keywords

IGNORE (program instruction)

LOCK (program instruction)

PRIORITY (real-valued function)

REACTE (program instruction)

REACTI (program instruction)

SIG.INS (real-valued function)

Syntax

```
REACTE program_name
```

Function

Initiate the monitoring of errors that occur during execution of the current program task.

Usage Considerations

The main purpose for the REACTE instruction is to allow for an orderly shutdown of the system if an unexpected error occurs. If a robot hardware error occurs, for example, a REACTE program can set external output signal lines to shut down external equipment. Using the REACTE instruction for other purposes requires extreme caution.

The REACTE instruction can be executed by any of the program tasks. That is, each task can have its own, independent REACTE definition. (A task cannot directly trap errors caused by another task, but tasks can signal each other via global variables or software signals.)

See the list below for other considerations.

Parameter

`program_name` Optional name of the program that is to be called when a program error occurs. If no program is specified, the previous REACTE is canceled, and any pending error message is discarded.

Details

If an error occurs after a REACTE instruction has been executed, the specified program will be invoked, rather than stopping normal program execution. (The program is invoked as though by the instruction CALL program.) The ERROR real-valued function can be used within the error-handling program to determine what error caused the program to be invoked.

There are several special considerations that must be kept in mind when using this facility:

- The program priority is raised to 254 when the error-handling program is invoked, locking out all reaction programs.
- Execution of the program task will stop if an error occurs while the system is processing a previous error.

- There must be room on the user program stack for one more subroutine. Therefore, the error *Too many subroutine calls* cannot be processed. (See the STACK monitor command.)
- The error-handling program can contain a RETURN instruction. When it is executed, the program will try to reexecute the instruction that caused the error. Note that this could cause an endless loop if the error continues to occur.
- If the reaction program exits at a RETURNE instruction, execution of the last-suspended program will resume at the step *following* the instruction that caused the program to be invoked.
- Once triggered (by an error), error processing is automatically disabled. Thus, another REACTE instruction must be executed if error handling is to continue.
- Before the error-handling program is entered, a DETACH instruction for the robot (logical unit number 0) is effectively executed. Thus, an ATTACH instruction must be executed for the robot before program control of the robot can resume.
- If a STOP, HALT, or PAUSE instruction is executed within the error-handling program, the original error message will be output unless the error-handling program contains a REACTE instruction with no argument.
- Unlike REACT and REACTI, execution of the REACTE error-handling program is never deferred because of priority considerations.

Example

Initiate monitoring of errors so that the program error.trap will be executed if any error should occur during execution of the current program task:

```
REACTE error.trap
```

Related Keywords

ERROR (real-valued function)

REACT (program instruction)

REACTI (program instruction)

RETURNE (program instruction)

Syntax

```
REACTI signal_num, program, priority
```

Function

Initiate continuous monitoring of a specified digital signal. Automatically stop the current robot motion if the signal transitions properly and optionally trigger a subroutine call.

Usage Considerations

For most applications, the REACTI instruction should be used only in a robot control program. (See below for more information.)

When a REACTI triggers, the robot that is stopped is the one selected by the task at the time of the trigger, regardless of which robot was selected at the time the REACTI instruction was executed.

Also see the considerations listed for the REACT program instruction.

Parameters

`signal_num` Real-valued expression representing the signal to be monitored. The signal number must be in the range 1001 to 1012 (external input signals) or 2001 to 2008 (internal software signals). (The software signals can thus be used by a secondary program to interrupt the robot control program, and vice versa.)

If the signal number is positive, V+ looks for a transition from off to on; if signal is negative, V+ looks for a transition from on to off.

`program` Optional name of the subroutine that is called when the signal transitions properly.

`priority` Optional real-valued expression that indicates the relative importance of this reaction as explained below. The value of this expression is interpreted as an integer value and can range from 1 to 127. If this argument is omitted, it defaults to 1. See the LOCK instruction for additional details on priority values.

Details

When the specified signal transition is detected, V+ reacts by immediately stopping the current robot motion. If a program is specified, V+ then continues processing the reaction just as it would for a REACT instruction. (See the description of the REACT instruction for a full explanation of this processing).

When REACTI is used by a program task that is not actually controlling the robot, care must be exercised to make sure the robot control program does not nullify the intended effect of the reaction subroutine. That is, if your application has one program task monitoring the signal and a different program task controlling the robot, you should keep the following points in mind when planning for processing of the reaction:

- The robot motion in process at the time of the reaction will be stopped, as if a BRAKE instruction were executed, but execution of the robot control program will not be directly affected.
- If a reaction subroutine is specified, that routine will be executed by the task that is monitoring the reaction (not by the task controlling the robot).

The signal monitoring continues until one of the following occurs:

- An IGNORE instruction is executed for the signal.
- A reaction occurs (in which case IGNORE signal_number is automatically performed).
- A REACTI (or REACT) instruction is executed that refers to the same signal. That is, if the signal specified in a REACTI instruction is already being monitored by a previous REACTI or REACT instruction, the old instruction will be canceled when the new REACTI instruction is executed.

If you do not want the robot motion to stop until the reaction program is actually called, you should use a REACT instruction and put a BRAKE instruction in the reaction program.

Example

The instruction below initiates monitoring of external input signal #1001. The robot motion will be stopped immediately if the signal ever changes from on to off (since the signal is specified as a negative value). A branch to program alarm will then occur when the program priority falls below 10 (if it is not already at or below that level).

```
REACTI –1001, alarm, 10
```

Related Keywords

IGNORE	(program instruction)
LOCK	(program instruction)
PRIORITY	(real-valued function)
REACT	(program instruction)
REACTE	(program instruction)
SIG.INS	(real-valued function)

Syntax

```
READ (lun, record_num, mode) var_list
```

Function

Read a record from an open file or from an attached device that is not file oriented. For an AdeptNet device, read a string from an attached and open TCP connection.

Usage Considerations

The logical unit referenced by this instruction must have been attached previously.

For file-oriented devices, a file must already have been opened with an FOPEN_ instruction.

The AdeptNet features of this instruction apply only to Adept MV controllers with the AdeptNet Ethernet option and the AdeptTCP/IP Protocol Access option license.

Parameters

lun Real-valued expression that identifies the device to be accessed. (See the ATTACH instruction for a description of unit numbers.)

record_num Optional real-valued expression that specifies the record to read for file-oriented devices opened in random-access mode (see the FOPEN_ instructions). For nonfile-oriented devices or for sequential access of a file, this parameter should be 0 or omitted. Records are numbered from one to a maximum of 16,777,216.

When accessing the TCP device with a server program, this parameter is an optional real variable that *returns* the client handle number. The handle can be used to identify the client accessing a multiple-client server.

`mode` Optional real-valued expression that specifies the mode of the read operation. Currently, the mode is used only for the terminal and serial I/O logical units. The value is interpreted as a sequence of bit flags as detailed below. (All bits are assumed to be clear if no mode value is specified.)

Bit 1 (LSB) Wait (0) vs. No-wait (1) (mask value = 1)

If this bit is clear, program execution is suspended until the read operation is completed. If the bit is set and the requested data is not available, program execution continues immediately and the IOSTAT function returns the error code for *No data received* (–526).

NOTE: For a no-wait READ to access a serial line, the line must be configured to use DDCMP.

Bit 2 Echo (0) vs. No-echo (1) (mask value = 2)

If this bit is clear, input from the terminal is echoed back to the source. If the bit is set, characters are not echoed back to the source. (This mode bit is ignored for the serial lines.)

`var_list` Either a list of real-valued input variables or a list of string variables, which will receive the data (see following details).

Details

This is a general-purpose data input instruction that reads a record from a specified logical unit. A record can contain an arbitrary list of characters but must not exceed 512 characters in length. For files that are opened in fixed-length record mode, this instruction continues to read characters until it has read exactly the number of characters specified during the corresponding FOPEN_ instruction.

For variable-length record mode (with most devices), this instruction reads characters until the first carriage-return (CR) and line-feed (LF) character sequence (or Ctrl+Z) is encountered. Thus, for example, if you perform a variable-length record mode read from the disk, you receive all the characters until a CR and LF are encountered.

The special character Ctrl+Z (26 decimal) indicates the logical end of the file, which is reported as an error by the IOSTAT function. No input characters can be read beyond that point.

READ operations from the terminal, the manual control pendant, and the serial lines are always assumed to be in variable-length record mode. Except as noted below, the records are terminated by CR and LF (which are not returned as part of the record). Thus, a READ from these devices will not be complete until a CR and LF are received as input. For example, if you perform a READ from the terminal, you receive all the characters until the RETURN key is pressed.[1]

> **NOTE:** The GETC real-valued function can be used instead of the READ instruction if you want to receive the CR and LF characters at the end of a record.
>
> When a READ instruction accesses a serial line configured to use DDCMP, the record may contain arbitrary data, including CR and LF characters.

If bit 1 is set in the `mode` value, a read operation that is not complete will not cause the program to wait, but will return immediately with the error ∗No data received∗ (error code –526). Then, additional READ instructions must be executed, until one is complete, in order to obtain the data in the variable list. The IOSTAT function can be used to determine when such a READ is complete.

Once a record has been read, it is processed in one of the following two ways:

1. If the **`var_list`** parameter is a list of *real-valued* variables, the record is assumed to contain a list of numbers separated by space characters and/or commas. Each number is converted from text to its internal representation, and its value is stored in the next variable contained in the variable list. If more values are read than the number of variables specified, the extra values are ignored. If fewer values are read, the remaining variables are set to zero. If data is read that is not a number, an error occurs and program execution stops.

[1] When a CR is received from the system terminal, V+ automatically adds a LF. Similarly, the pendant's DONE key is interpreted as CR and LF.

2. If the **var_list** parameter is a list of string variables, the entire record is stored in the string variables as follows. The first 128 bytes in the record are copied to the first string variable. If there are more than 128 bytes in the record, the second string variable is filled with the next 128 bytes. This continues until the entire record has been processed or all the string variables have been filled.

   If there is not enough data to fill all the string variables, the unused string variables will be set to the empty string (). If there is too much data for the number of string variables specified, an error will be reported by IOSTAT.

   When a READ is performed in variable-length record mode, the strings will contain all the characters up to, but not including, the terminating CR and LF, which are discarded.

Any error in the specification of this instruction (such as attempting to read from an invalid unit) will cause a program error and will halt program execution. However, errors associated with performing the actual read operation (such as end of file or device not ready) will not halt program execution since these errors may occur in the normal operation of a program. These normal errors can be detected by using the IOSTAT function after performing the read. In general, it is good practice always to test whether each read operation completed successfully by testing the value from IOSTAT.

When accessing the AdeptNet device, the `record_num` parameter allows a server to communicate with multiple clients on a single logical unit. In this context, the parameter provides a *handle* number that you can use to identify the client from which the READ data was received. Handles are allocated when a client connects to the server and then are deallocated when the client disconnects. In order to determine when the client connection or disconnection is done, you must use the IOSTAT real-valued function after the READ. Refer to the documentation for IOSTAT.

The READ instruction with TCP/IP reads data until either the input string is full or the buffer is empty, at which point the instruction returns. READ with TCP/IP does not allow fixed-length records and does not terminate when encountering a delimiter.

Example

Read a line of text from the disk and store the record in the string variable **$disk.input**:

```
READ (5) $disk.input
```

For an example of using the READ instruction with the TCP device, refer to the Example section for the IOSTAT real-valued function.

Related Keywords

ATTACH (program instruction)

FOPEN_ (program instruction)

FSEEK (program instruction)

GETC (real-valued function)

IOSTAT (real-valued function)

PROMPT (program instruction)

Syntax

READY

Function

Move the robot to the READY location above the workspace, which forces the robot into a standard configuration.

Usage Considerations

Before executing this instruction with the DO monitor command (DO READY), make sure that the robot will not strike anything while moving to the READY location.

The READY instruction can be executed by any program task so long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the READY instruction causes an error.

Details

This instruction will always succeed, regardless of where the robot is located at the time.

An Adept SCARA robot has the following configuration when it is at the READY location:

- Joint 2 is at 90 degrees
- The axis of joint 3 is in the World X-Z plane (that is, Y = 0)
- If the optional joint 5 is installed, it is pointed straight down (AdeptOne and AdeptThree only)
- The alignment keyway in the end-effector flange is directed along the positive X axis (that is, the tool X axis is parallel to the world X axis)

Chapter Table 2-18. lists the joint positions for the READY locations for the AdeptOne, PackOne, and AdeptThree SCARA robots and the Adept UltraOne Cartesian robot.

Table 2-18. Approximate Joint Positions for READY Location

Joint	AdeptOne Rt-handed	PackOne Rt-handed	AdeptThree Rt-handed
1	–41.4°	–45.0°	–42.3°
2	90.0°	90.0°	90.0°
3	25.0 mm	25.0 mm	25.0 mm
4	48.6°	45.0°	47.7°
5	0.0°	N.A.	0.0°

For devices controlled by the AdeptMotion VME, the READY location depends upon the device module being used.

Related Keyword

SELECT (program instruction and real-valued function)

Syntax

RELAX

RELAXI

Function

Limp the pneumatic hand.

Usage Considerations

RELAX causes the hand to limp during the next robot motion.

RELAXI causes a BREAK in the current continuous-path motion and causes the hand to limp immediately after the current motion completes.

The RELAX instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

The RELAXI instruction can be executed by any program task so long as the task has attached a robot. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing these instructions will cause an error.

Details

These instructions turn off both the open and close pneumatic control solenoid valves, causing the pneumatic hand to become limp. If the RELAX instruction is used, the signal will be sent when the next robot motion begins.[1]

The RELAXI instruction differs from RELAX in the following ways:

- A BREAK occurs if a continuous-path robot motion is in progress.
- The signals are sent to the control valves at the conclusion of the current motion or immediately if no motion is in progress.
- Robot motions are delayed for a brief time to allow the hand actuation to complete. The length of the delay (in seconds) is the current setting of the HAND.TIME system parameter.

[1] Use theSPEC Utility program to set the digital signals that control the pneumatic hand. See the *Instructions for Adept Utility Programs* for information on use of the program.

Related Keywords

CLOSE and **CLOSEI**	(program instructions)
HAND.TIME	(system parameter)
OPEN and **OPENI**	(program instructions)
SELECT	(program instruction and real-valued function)

Syntax

RELEASE `task`

Function

Allow the next available program task to run.

Parameter

`task` Optional argument specifying the task to release control to. The range of acceptable values is from –1 to the number of tasks configured for the current system. The default value is –1.

If the specified task does not exist, the current task resumes.

Details

If `task` is omitted, the instruction releases control to the highest priority task within the time slice that is ready to run and is not in the same round-robin group as the current task. See the ***V+ Language User's Guide***.

If `task` = –1, the instruction releases control to another task, and V+ scans as follows:

- If the current task is explicitly scheduled in this time slice, begin scanning with the next lower priority task in this slice. When the end of the slice is reached, wrap around to the beginning of this slice and search until the current task is found again. Ignore the current task and any task in the current task's round-robin group.
- If the current task is not scheduled in this slice, scan this slice from the top to the bottom. Ignore any task in the current task's round-robin group.
- If no ready task is found in the current slice, search ahead in all other slices until a task is found that is ready to run. Accept tasks in the current task's round-robin group. If no other ready task is found, resume the current task.

If `task` >= 0, control is released to the specified task, provided that task is ready to run. If the task is not ready to run, the current task continues executing. This release bypasses the normal time slice and priority checking.

> **NOTE:** If two high-priority tasks perform releases in the same time slice, they pass control back and forth to each other, effectively locking out any lower-priority tasks in the slice.

This instruction can be used in place of the WAIT instruction (with no arguments) in cases where other tasks must be given an opportunity to run, but a delay until the next 16-millisecond cycle is not desired.

Related Keyword

WAIT (program instruction)

Syntax

RESET

Function

Turn off all the external output signals.

Details

The RESET program instruction is useful in the initialization portion of a program to ensure that all the external output signals are in a known state.

WARNING: Before issuing this instruction, make sure all devices connected to the output signals can safely be turned off. Be especially careful of signals that start an action when they are turned off.

Related Keywords

BITS	(monitor command, program instruction, and real-valued function)
IO	(monitor command)
RESET	(monitor command)
SIG	(real-valued function)
SIG.INS	(real-valued function)
SIGNAL	(monitor command and program instruction)

See the *V+Operating System Reference Guide* for details on monitor commands.

Syntax

```
... RETRY
```

Function

Control whether the PROGRAM START button causes a program to resume.

Usage Considerations

This switch has no effect on systems that do not have a PROGRAM START button on the controller front panel.

Your controller must be attached to either an optional Adept front panel or a user supplied external front panel.

Details

When the RETRY switch is disabled, the controller front-panel PROGRAM START button will not resume execution of the robot control program. (See the description of the RETRY monitor command for all the conditions that must be satisfied in order to use the PROGRAM START button to invoke a RETRY command.)

This switch has no effect on the operation of the WAIT.START monitor command.

The RETRY switch is initially disabled.

Related Keywords

DISABLE	(monitor command)
ENABLE	(monitor command)
RETRY	(monitor command)
SWITCH	(monitor command, program instruction, and real-valued function)

Syntax

RETURN

Function

Terminate execution of the current subroutine, and resume execution of the last-suspended program at the step following the CALL or CALLS instruction that caused the subroutine to be invoked.

Details

A RETURN instruction in a main program has the same effect as a STOP instruction.

A RETURN instruction is assumed if program execution reaches the last step of a subroutine. However, it is not good programming style to use this feature—an explicit RETURN instruction should be included as the last line of each subroutine.

The effect of a RETURN instruction in an error reaction subroutine differs slightly. In that case, if the reaction subroutine was invoked because of a program error (as opposed to an asynchronous servo error or PANIC button press), the statement that caused the error will be executed again. That is, the error could occur again immediately. The RETURNE instruction should be used in error reaction subroutines to avoid that situation.

Related Keywords

CALL (program instruction)

CALLS (program instruction)

RETURNE (program instruction)

Syntax

RETURNE

Function

Terminate execution of an error reaction subroutine and resume execution of the last-suspended program at the step following the instruction that caused the subroutine to be invoked.

Details

The RETURNE instruction is intended for use in error reaction subroutines. That is, subroutines that are invoked, through the REACTE mechanism, as a result of an error during program execution.

When a RETURNE instruction is executed in an error reaction subroutine, then execution continues with the statement following the one executing when the error occurred. (Note that in this situation, a RETURN instruction would result in the statement that generated the error being executed again, possibly causing an immediate repeat of the error.)

> **NOTE:** Due to the forward processing ability of V+, the instruction that is the source of an error may not be the one executing when the error is actually registered. For example, when a MOVE instruction is processed, the robot begins moving, but during the motion several additional instructions may be processed. If an envelope or similar error occurs after this forward processing, the RETURNE will be based on the instruction processing when the error occurs, not the MOVE instruction.

It may be helpful to note that the RETURNE instruction behaves similarly to the PROCEED command. The RETURN instruction behaves similarly to the RETRY command (except that with RETURN an interrupted robot motion will not be restarted).

A RETURNE instruction in a program that is *not* executed in response to an error has the same effect as a RETURN instruction. RETURNE, however, takes slightly longer to execute than does RETURN.

Related Keywords

REACTE (program instruction)

RETURN (program instruction)

Syntax

RIGHTY

Function

Request a change in the robot configuration during the next motion so that the first two links of the robot resemble a human's right arm.

Usage Considerations

Configuration changes cannot be made during straight-line motions.

If the selected robot does not support a right-handed configuration, this instruction is ignored by the robot.

The RIGHTY instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the RIGHTY instruction will cause an error.

See **Figure 2-5 on page 403**.

Related Keywords

CONFIG	(real-valued function)
LEFTY	(program instruction)
SELECT	(program instruction and real-valued function)

Syntax

```
... ROBOT [index]
```

Function

Enable or disable one robot or all robots.

Usage Considerations

The ROBOT system switches may be modified only when both of the following conditions are satisfied:

1. The POWER system switch is OFF.
2. When the V+ system was booted from disk, at least one robot started up without reporting a fatal error.

Parameter

`index`	Optional real value, variable, or expression (interpreted as an integer) that specifies the robot to be enabled or disabled. The value should be 1, 2, or 3 (corresponding to robots 1, 2, or 3, respectively). All three elements of the ROBOT switch are modified when this parameter is omitted.

Details

When the V+ system starts up (after booting from disk), all the robots that started up without reporting a fatal error are enabled by default, and all the corresponding ROBOT switches are enabled. After start-up, the ROBOT switches can be used to selectively disable robots. For example, this can aid in the debugging of individual robots.

The ROBOT switches may be modified only for robots that are present, and that started up without a fatal error.

When a robot is disabled by use of the ROBOT switch, that robot will be bypassed when:

- Power is enabled for all robots with the COMP/PWR button on the manual control pendant, or with the POWER system switch.
- All the robots are calibrated via the CALIBRATE monitor command or program instruction.

Motion instructions should not be executed for a robot that has been disabled.

The settings of these switches can be checked at any time with the SWITCH monitor command or real-valued function to determine which robots are enabled.

Related Keywords

DISABLE	(monitor command and program instruction)
ENABLE	(monitor command and program instruction)
SWITCH	(monitor command, program instruction, and real-valued function)

Syntax

```
ROBOT.OPR (function_code) value, value, ..., value
```

Function

Execute operations that are specific to the currently selected robot or robot module.

Usage Considerations

ROBOT.OPR is a general-purpose instruction whose interpretation varies from one robot type to another.

Parameters

`function_code`	Optional real value that specifies a function for the selected robot module.
`value`	Optional expression whose interpretation is determined by the selected robot module.

Details

This instruction executes operations that are specific to the currently selected robot or robot module. If the selected robot does not support any special operations, this instruction has no effect. Currently, it applies only to the X/Y/3(Z/Theta) robot module. For that module `function_code` = 0 enables selection of the primary and slaved axes, and `function_code` = 1 enables definition of the position-offset values that are added to the commanded Z and RZ positions for each slave axis. For details about the applicability and use of this instruction, refer to the documentation for your specific robot module.

Syntax

```
RUNSIG signal_num
```

Function

Turn on (or off) the specified digital signal as long as execution of the invoking program task continues.

Usage Considerations

Only one RUNSIG signal can be in effect for each program task.

Parameter

`signal_num` Optional real-valued expression that specifies one of the digital output signals (or an internal software signal) that is to be controlled.

The signal will be set to on during program execution if the value is positive. A negative value will result in the signal being set to off during program execution, and turned on when execution stops.

If no signal is specified, any RUNSIG in effect for the task will be canceled.

Details

This instruction causes the specified digital signal to be turned on (or off) as soon as the instruction is executed. The signal will be turned off (or on) as soon as execution of the invoking program task stops (or the STOP instruction is executed).

This instruction is useful in an application where auxiliary equipment must be stopped if an error occurs during program execution.

Only one signal can be activated by a RUNSIG instruction at any one time (for each program task). An error condition will result unless a program cancels the first RUNSIG before attempting to initiate a second.

If program execution is interrupted after a RUNSIG instruction has been executed, the specified signal will return to the selected state again if a PROCEED or RETRY command is issued. If an SSTEP or XSTEP command is issued, the signal will return to the specified state during execution of the instruction that is invoked. Similarly, processing of a DO command temporarily activates the RUNSIG signal for the corresponding program task. (The EXECUTE command and instruction cancel any previous RUNSIG for the specified program task.)

Example

```
RUNSIG run.signal
```

Turns on the digital signal identified by the value of the variable run.signal (assuming the value is positive). The signal will remain on throughout execution of the current program. The signal will go off when execution ends.

Related Keywords

IO	(monitor command)
RESET	(monitor command)
SIG	(real-valued function)
SIG.INS	(real-valued function)
SIGNAL	(monitor command and program instruction)

See the ***V+ Operating System Reference Guide*** for details on monitor commands.

Syntax

```
RX (angle)

RY (angle)

RZ (angle)
```

Function

Return a transformation describing a rotation.

Parameter

angle	Real-valued expression that represents the rotation angle in degrees.

Details

These functions generate a transformation whose value consists of a rotation about the axis associated with the function name and a zero displacement (X, Y, Z = 0).

Example

RX(30)	Produces a transformation that describes a pure 30-degree rotation about the World X axis.

Related Keyword

DX DY DZ	(real-valued functions)

Syntax1

```
SCALE (transformation BY scale_factor)
```

Function

Return a transformation value equal to the transformation parameter with the position scaled by the scale factor.

Parameters

`transformation` Transformation expression that is to be scaled.

`scale_factor` Real-valued expression that is used to scale the transformation parameter value.

Details

The value returned is equal to the value of the input transformation parameter value except that the X, Y, and Z position components are multiplied by the scale factor parameter. The rotation components have their values unchanged.

Example

If the transformation x has the value:

```
(200, 150, 100, 10, 20, 30)
```

then executing the instruction:

```
SET y = SCALE(x BY 1.25)
```

will result in the transformation y receiving the value:

```
(250, 187.5, 125, 10, 20, 30)
```

Related Keyword

SHIFT (transformation function)

Syntax

```
... SCALE.ACCEL [robot_num]
```

Function

Enable or disable the scaling of acceleration and deceleration as a function of program speed when program speed is below a preset value.

Parameter

`robot_num`	Optional real value, variable, or expression (interpreted as an integer) that indicates the number of the robot affected.

Details

If `robot_num` is omitted or zero in an ENABLE or DISABLE command or instruction, the settings for all robots are altered. Otherwise, only the setting for the specified robot is affected. If `robot_num` is omitted or zero when the switch is accessed with the SWITCH real-valued function, the setting of the switch for robot #1 is returned.

With the SPEC utility you can set the speed limit below which the acceleration and deceleration rates are scaled. If the threshold parameter is set to 100% or greater speed, enabling SCALE.ACCEL has no effect. When this switch is enabled, all robot modules have the SCALE.ACCEL speed limit set by default to a very large value, effectively forcing the scaling of accelerations for all speeds.

When this switch is enabled and the program speed is below the preset value, the effective acceleration or deceleration is calculated as follows:

- Effective acceleration = program_speed * acceleration
- Effective deceleration = program_speed * deceleration

  where the acceleration and deceleration are values set by the ACCEL instruction.

This switch is enabled when the V+ system is initialized.

CAUTION: For program speeds over 100%, if the default setting for the SCALE.ACCEL limit is used and SCALE.ACCEL is enabled, the robot is driven at much higher rates of acceleration and deceleration, as compared to V+ 11.0.

Example

```
DISABLE SCALE.ACCEL[2]
;Turn off acceleration scaling for robot #2
```

Related Keywords

ACCEL (program instruction and real-valued function)

SPEED (monitor command and program instruction)

Syntax

```
... SCALE.ACCEL.ROT [robot_num]
```

Function

Specify whether or not the SCALE.ACCEL switch takes into account the Cartesian rotational speed during straight-line motions.

Parameter

`robot_num`	Optional real value, variable, or expression (interpreted as an integer) that indicates the number of the robot affected.

Details

Prior to V+ version 11.3, during straight-line motions the lesser of the Cartesian linear and rotational speeds was used to scale acceleration and deceleration. If SCALE.ACCEL.ROT is enabled for a selected robot, the effect of SCALE.ACCEL is the same as in previous versions of the V+ language. However, if SCALE.ACCEL.ROT is disabled for a selected robot, only the Cartesian linear speed is considered when SCALE.ACCEL is in effect. The SCALE.ACCEL.ROT switch is enabled for all robots by default when the V+ system is initialized.

Example

```
DISABLE SCALE.ACCEL.ROT[2] ;Cause SCALE.ACCEL not to use
                           ;Cartesion rotational speed
                           ;for robot #2
```

Related Keywords

ACCEL	(program instruction and real-valued function)
SPEED	(monitor command and program instruction)

Syntax

```
... SCREEN.TIMEOUT
```

Function

Establish the time-out period for blanking the screen of the graphics monitor.

Usage Considerations

The SCREEN.TIMEOUT system parameter is ignored by Adept S-series controllers.

The current value of the SCREEN.TIMEOUT parameter can be determined with the PARAMETER monitor command or real-valued function.

The value of the SCREEN.TIMEOUT parameter can be modified only with the PARAMETER monitor command or program instruction.

The parameter name can be abbreviated.

Details

The SCREEN.TIMEOUT parameter sets the number of seconds of inactivity it takes to trigger screen blanking. That is, when the pointing device, keyboard, and graphics screen have been idle for the specified time, the monitor screen is blanked.

The screen is refreshed at the next occurrence of any of the following events:

- The pointer device is moved
- Any key on the keyboard is pressed.
- Any graphics or text is written to the screen.
- Any button on the pointer device is clicked.

> **NOTE:** Because the system can be waiting for input from the user, take care when waking up the screen. For example, pressing the SHIFT key or moving the mouse does not actually generate any input, and thus they are safe to use to restore the screen.

The value of this parameter is interpreted as an integer. It can range from 0 to 16383, inclusive. Screen blanking is disabled if this parameter is set to 0.

This parameter is set to 0 when the V+ system is initialized.

Example

Set the time-out limit to 600 seconds (10 minutes):

```
PARAMETER SCREEN.TIMEOUT = 600
```

Related Keyword

PARAMETER (monitor command, program instruction & real-valued function)

Syntax

```
SEE (lun) prog_spec, step
```

Function

Invoke the screen-oriented program editor to allow a program to be created, viewed, or modified.

Usage Considerations

Programs that are being edited in read-write mode cannot be executed. If an active program attempts to CALL a program that is being thus edited, the executing program will be terminated with an error.

Programs that are being executed cannot be edited in read-write access mode. This restriction also applies to suspended programs in the stack of an executing program task.

Programs that are executing and read-only programs are automatically edited in read-only mode. In that mode programs can be viewed, but no commands are accepted that would modify the program. Protected programs cannot be edited at all. (See the DIRECTORY command for an explanation of protected and read-only programs.)

Program output to the specified logical unit is blocked during an edit session (see below).

The editor invoked with this instruction does not access (or affect) the user-specified information retained by the command SEE editor.

Parameters

lun Real value, variable, or expression (interpreted as an integer) that identifies the serial line connected to the terminal to be used for editing, or the graphics window to be used for editing. (See the ATTACH instruction for a description of unit numbers.)

NOTE: The logical unit must have been attached by the program.

If a graphics window is specified, the window must be open and must satisfy the size constraints described below.

prog_spec	String variable, constant, or expression that specifies the name of the program to be edited. Unlike the SEE monitor command, a program name must be specified. If the program is to be accessed in read-only mode, the string specifying the program name must end with /R.
step	Optional number specifying the program step at which editing is to begin. If the step number is omitted, editing will begin at the second step of the program.

Details

This instruction is used to start an editing session with the V+ screen editor. The instruction SEE editor can be used at the same time as the command SEE editor. Thus, for example, it is possible for more than one person to be editing programs on the V+ system at the same time.

Details of the SEE editor, including descriptions of all the commands it accepts, are presented in the ***V+ Language User's Guide***.

The SEE program instruction differs from the SEE monitor command in the following ways:

1. For a system that has a graphics system processor, the SEE monitor command always initiates editing with the Monitor window and the system keyboard. The SEE program instruction can be used to initiate editing of a program in a different graphics window, or with a terminal connected to one of the USER serial ports.
2. For a system that does not have a graphics system processor, the SEE monitor command always initiates editing with the system terminal. The SEE program instruction can be used to initiate editing of a program with a terminal connected as the system terminal or to one of the USER serial ports.
3. A program name must be specified in the SEE instruction.
4. A step number can be specified with the SEE program instruction.
5. The program debugger cannot be invoked during a session invoked with the SEE instruction.
6. When initiated with the SEE program instruction, user-specified settings (such as extended-command settings and macros) are *not* retained between editing sessions. (User-specified settings entered during a session initiated with the SEE instruction do not affect the corresponding information accessed by the SEE monitor command.)

If the **lun** parameter refers to a graphics window, the window must be at least 7 lines (105 pixels) high, and cannot be higher than 64 lines (960 pixels). The window must be 82 characters (656 pixels) wide.

Related Keywords

DEBUG	(monitor command)
EDIT	(monitor command)
SEE	(monitor command)

See the *V+ Operating System Reference Guide* for details on monitor commands.

Syntax

```
SELECT device_type = unit
```

Function

Select a unit of the named device for access by the current task.

Usage Considerations

The SELECT instruction needs to be used only if there are multiple devices of the same type connected to your system controller. This option is available only if your Adept system is equipped with the V+ Extensions option.

The SELECT instruction affects only the task in which the instruction is executed.

The instruction SELECT ROBOT can be executed only if there is no robot attached to the current task. (If there is any doubt about whether or not a robot is attached, a program should execute a DETACH instruction before executing the SELECT instruction.)

Parameters

`device_type` Keyword that identifies the type of device that is to be selected. Valid device types are ROBOT, VISION, and FORCE (which must be specified without quotation marks). The device-type keyword can be abbreviated.

`unit` Real value, variable, or expression (interpreted as an integer) that specifies the particular unit to be selected. The values that are accepted depend on the configuration of the system.

Details

SELECT ROBOT

In a multiple-robot system, this program instruction selects the robot with which the current task is to communicate. (The SELECT monitor command specifies which robot the V+ monitor is to access.) The program instruction specifies which robot will receive motion instructions (for example, APPROACH and MOVE) and will return robot-related information (for example, for the HERE function).

Each time a program task begins execution, robot #1 is automatically selected. If a robot is selected, information about the robot (for example, its current position) can be accessed. In order for a program to move a robot, however, the robot must be selected and attached (with the ATTACH instruction).

As an example, if robot #2 is selected by a SELECT instruction, all motion instructions executed by the current task will be directed to that robot (until another SELECT instruction is issued). Also, all robot-related functions (such as HERE) will return information about robot #2.

> **NOTE:** As a convenience, when task #0 is executed, robot #1 is automatically selected and attached when program execution begins.

In order for any task to change its selected robot, no robot can be attached by the task. More than one task can have a particular robot selected, but only one task can have a robot attached. If a robot is already attached to a different task, an ATTACH will wait or generate an error (depending on the mode parameter for the ATTACH instruction).

SELECT VISION

In a system with multiple vision systems, this instruction selects the vision system with which the current task is to communicate. (The SELECT monitor command specifies which vision system the V+ monitor is to access.) This program instruction specifies which vision system will receive vision instructions (for example, ENABLE VISION) and also which system will return vision-related information (for example, from the VSTATUS function).

Vision system #1 is automatically selected each time a program begins execution.

SELECT FORCE

In a system with multiple force sensors, this monitor command or program instruction selects the force sensor with which the current task is to communicate. The SELECT monitor command specifies which force sensor the V+ monitor is to access. The program instruction specifies which force sensor will receive force instructions (for example, FORCE.READ) and will return force sensor-related information (for example, for the LATCH function).

Each time a program task begins execution, force sensor #1 is automatically selected.

Example

SELECT ROBOT

The following program selects robot #3 and moves it. This program would normally not be executed by task #0, since that task is attached to robot #1 by default.

```
.PROGRAM test()

   SELECT ROBOT = 2 ;Select robot 3

   ATTACH (0,1)     ;Get control of robot 3 without waiting
   IF IOSTAT(0) < 0 THEN
     TYPE /B, "Error attaching robot: ", $ERROR(IOSTAT(0))
     PAUSE
   END

   MOVE x           ;Move robot 3 to location "x"
   MOVE y           ;Move robot 3 to location "y"

   DETACH           ;Detach robot 3

.END
```

SELECT VISION

The following program segment selects vision system #2 and accesses that system.

```
PROGRAM vision.2()
   SELECT VISION = 2     ;Select vision system #2
   ENABLE VISION         ;Enable that vision system
   VSTATUS(1,0) status[] ;Get status information
   IF status[0] == 0 THEN;If vision system is idle,
   VPICTURE(1)           ;take a picture
END
```

SELECT FORCE

The following program selects force sensor #2 and reads the current forces from it.

```
.PROGRAM test()

   SELECT FORCE = 2     ;Select force sensor 2
   FORCE.READ f[]       ;Read sensor 2 forces
   TYPE Current forces on sensor, SELECT(FORCE), /S
   TYPE  are , /F0.1, f[0], /X1, f[1], /X1, f[2]

.END
```

Related Keywords

ATTACH (program instruction)

SELECT (real-valued function)

Syntax

SELECT (device_type, mode)

Function

Return the unit number that is currently selected by the current task for the device named.

Parameter

device_type Keyword that identifies the type of device that is to be selected. Valid device types are ROBOT, VISION, and FORCE (which must be specified without quotation marks). The device-type keyword can be abbreviated.

mode Optional real value, variable, or expression (interpreted as an integer) that specifies the mode for the function. If this parameter is omitted or has the value 0, the function returns the number of the unit currently selected, or 0 if no unit is selected. If mode has the value –1, the function returns the total number of units available for the specified device.

Details

This function returns either the number of the specified device that is currently selected, or the total number of devices connected to the system controller. Multiple devices of the same type are supported only if your system includes the optional V+ Extensions software.

If the V+ system is not configured to control a robot, the selected robot is always #1, and the total number of robots is zero.

SELECT(ROBOT) returns the number of the currently selected robot.
SELECT(ROBOT,–1) returns the maximum robot number in a V+ system.[1]
SELECT(VISION) returns the number of the currently selected vision system.
SELECT(FORCE) returns the number of the currently selected force sensor.

A particular FORCE sensor is associated with one VFI (VME Force Interface), which in turn has a number associated with it through the system configuration. With the default configuration, FORCE sensor number 1 corresponds to the VFI configured as servo board number 8.

[1] The older syntax, SELECT(ROBOT,1), is still supported but may be made obsolete in the future.

Examples

```
;Return the unit number of the robot selected for the current
;task

our.robot = SELECT(ROBOT)

;Return the unit number of the force sensor selected for the
;current task

our.force = SELECT(FORCE)

;Return the total number of robots connected to the
;controller

num.robots = SELECT(ROBOT,-1)

;Return the total number of vision systems installed in the
;controller

num.vision = SELECT(VISION,-1)
```

Related Keyword

SELECT (monitor command and program instruction)

Syntax

```
SET location_var = location_value
```

Function

Set the value of the location variable on the left equal to the location value on the right of the equal sign.

Parameters

location_var Single location variable or compound transformation that ends with a transformation variable.

location_value Location value of the same type as the location variable on the left of the equal sign, defined by a variable or function (or compound transformation).

Details

An error message is generated if the right-hand side is not defined or is not the same type of location representation (that is, transformation or precision point).

If a compound transformation is specified to the left of the equal sign, only its right-most relative transformation is defined. An error condition results if any other transformation in the compound transformation is not already defined.

If a transformation variable is specified on the left-hand side, the right-hand side can contain a transformation, a compound transformation, or a transformation function.

Examples

Set the value of the transformation pick equal to the location of corner plus the location of shift relative to corner:

```
SET pick = corner:shift
```

Set the value of the precision point #place equal to that of the precision point #post:

```
SET #place = #post
```

Set the value of loc1 to X = 550,Y = 450, Z = 750, y = 0, p = 180, r = 45:

```
SET loc1 = TRANS(550, 450, 750, 0, 180, 45)
```

Related Keywords

HERE	(monitor command and program instruction)
POINT	(monitor command, see the ***V+ Operating System Reference Guide)***

Syntax

```
SET.EVENT task, flag, processor
```

Function

Set an event associated with the specified task.

Parameters

task	Real value, variable, or expression (interpreted as an integer) that specifies the task for which the event is to be set. The valid range is 0 to 27, inclusive.[1]
flag	Not used, defaults to 1.
processor	Optional real value, variable, or expression that specifies the V+ processor running the task to be signaled. The default value is 0 which indicates the local processor. The maximum value depends on the software and hardware configuration. (This option is available only if your Adept system is equipped with the V+ Extensions option.)

Details

This instruction sets the event associated with the specified task. For example, if a task had been suspended by a WAIT.EVENT 1 instruction, executing the SET.EVENT instruction for that task will cause it to resume execution (during the next available time slice for which it is eligible).

Related Keywords

CLEAR.EVENT (program instruction)

GET.EVENT (real-valued function)

INT.EVENT (program instruction)

WAIT.EVENT (program instruction)

[1] All 28 tasks are available only in systems equipped with the optional V+ Extension.

Syntax

#SET.POINT

Function

Return the commanded joint-angle positions computed by the trajectory generator during the last trajectory-evaluation cycle.

Details

For each trajectory-evaluation cycle, joint-angle positions are computed, converted to encoder counts, and sent to the servos as the commanded motor positions. You can use this function to capture these positions.

Syntax

```
... SET.SPEED
```

Function

Control whether or not the monitor speed can be changed from the manual control pendant. The monitor speed cannot be changed when the switch is disabled.

Details

Priming a program from the manual control pendant normally sets the monitor speed. The speed is not changed when the SET.SPEED switch is disabled. Therefore, the operator cannot affect the program speed from the pendant.

Related Keywords

SPEED	(monitor command, see the *V+ Operating System Reference Guide)*
DISABLE	(monitor command and program instruction)
ENABLE	(monitor command and program instruction)
SWITCH	(monitor command, program instruction, and real-valued function)

Syntax

```
SETBELT %belt_var = expression
```

Function

Set the encoder offset of the specified belt variable equal to the value of the expression.

Usage Considerations

This option is available only if your Adept system is equipped with the V+ Extensions option.

The BELT switch must be enabled for this instruction to be executed.

SETBELT cannot be executed while the robot is moving relative to the specified belt variable.

The belt variable referenced must have been defined already using a DEFBELT instruction.

Parameters

%belt_var	Name of belt variable associated with the encoder offset to be set.
expression	Real-valued expression that specifies a signed 24-bit encoder offset value.

Details

When computing the position of a belt associated with a belt variable, V+ subtracts the offset value from the current belt position value and uses the difference, modulo 16,777,216.

The expression value is normally a signed number in the range –8,388,608 to 8,388,607. If the number is outside this range, its value modulo 16,777,216 is used.

Frequently, SETBELT is used in conjunction with the BELT real-valued function to set the effective belt position to zero.

For systems with the AdeptVision VME option, the SETBELT instruction can be used to synchronize robot motion with the encoder value reported by the vision system (via the VFEATURE real-valued function).

Example

The following example waits for a digital signal and then sets the belt position to zero. That is done by setting the belt offset equal to the current belt position. Finally, the robot is moved onto the belt.

```
WAIT sig(1001)
SETBELT %belt1 = BELT(%belt1)
MOVES %belt1:pickup
```

Related Keywords

BELT (system switch and real-valued function)

DEFBELT (program instruction)

WINDOW (program instruction and real-valued function)

Syntax

```
SETDEVICE (type, unit, error, command) p1, p2, ...
```

Function

Initialize a device or set device parameters. (The actual operation performed depends on the device referenced.)

Usage Considerations

The syntax contains optional parameters that apply only to specific device types and commands.

Parameters

type	Real value, variable, or expression (interpreted as an integer) that indicates the type of device being referenced.
unit	Real value, variable, or expression (interpreted as an integer) that indicates the device unit number. The value must be in the range 0 to (max –1), where max is the maximum number of devices of the specified type. The value should be 0 if there is only one device of the given type.
error	Optional real variable that receives a standard system error number that indicates if this instruction succeeded or failed. If this parameter is omitted, any device error stops program execution. If an error variable is specified, the program must explicitly check it to detect errors.
command	Real value, variable, or expression that specifies which device command or parameters are being set by this instruction. Some commands are standard and recognized by all devices. Other commands apply only to particular device types.
p1, p2, ...	Optional real values, variables, or expressions, the values of which are sent to the device as data for a command. The number of parameters specified and their meanings depend upon the particular device type being accessed.

Details

SETDEVICE is a general-purpose instruction for initializing external devices. It initializes the software and allows various parameters associated with the device to be set.

Two standard SETDEVICE commands are recognized by all devices:

command = 0 **Initialize device—**This command should be issued once before accessing the device with any other command. Normally, no additional parameters are required, but some device types may permit them.

command = 1 **Reset device—**This command resets the device. Normally no additional parameters are required, but some device types may permit them.

See the supplementary documentation for specific devices for details and examples.

The *V+ Language User's Guide* contains information on use of the SETDEVICE instruction to access external encoders.

Related Keywords

DEVICE (program instruction and real-valued function)

DEVICES (program instruction)

Syntax

SHIFT (transformation BY x_shift, y_shift, z_shift**)**

Function

Return a transformation value resulting from shifting the position of the transformation parameter by the given shift amounts.

Parameters

transformation	Transformation expression that is to be shifted. Optional real-valued expressions that are added to the respective position components of the transformation parameter.
x_shift	
y_shift	
z_shift	

Details

The value returned is equal to the value of the input transformation parameter value except that the three shift parameter values are added to the X, Y, and Z position components. If any shift parameter is omitted, its value is assumed to be zero.

Example

If the transformation x has the value:

```
(200, 150, 100, 10, 20, 30)
```

then executing the instruction:

```
SET y = SHIFT(x BY 5,-5,10)
```

will result in the transformation y receiving the value:

```
(205, 145, 110, 10, 20, 30)
```

Related Keywords

SCALE (transformation function)

TRANS (transformation function)

Syntax

```
SIG (signal_num, ..., signal_num)
```

Function

Returns the logical AND of the states of the indicated digital signals.

Parameter

signal_num Real-valued expression that evaluates to a digital I/O or internal signal number. A negative value indicates negative logic for that signal.

Details

Returns a TRUE (–1) or FALSE (0) value obtained by performing a logical AND of the states of all the indicated digital signals. That is, SIG will return TRUE if all the specified signal states are TRUE. Otherwise, SIG will return FALSE.

The magnitude of each **signal_num** parameter determines which digital or internal signal is to be considered. Signals 1 - 8 and 33 - 512 are digital outputs. Signals 1001 - 1012 and 1033 - 1512 are digital inputs. Signals 2001 to 2512 are internal (software) inputs or outputs. Only digital signals that are actually installed can be used. You can use the IO monitor command (or the SIG.INS function) to check your current digital I/O configuration.

If the sign of a **signal_num** parameter is positive, the signal is interpreted as being TRUE if it has a high value. If the sign of a **signal_num** parameter is negative, the signal is interpreted as being TRUE if it has a low value.

NOTE: SIG(0) returns a value of TRUE.

Example

Assume that the following digital I/O signals are installed and have the indicated values.

- Input signal 1001 is On
- Input signal 1004 is Off
- Input signal 33 is Off

The following SIG function references will then return the indicated values:

```
SIG(1001)         ;Returns -1.0 (TRUE)
SIG(1004)         ;Returns  0.0 (FALSE)
SIG(-1004)        ;Returns -1.0 (TRUE)
SIG(1001,1004)    ;Returns  0.0 (FALSE)
SIG(1001,-1004)   ;Returns -1.0 (TRUE)
```

Related Keywords

BITS	(monitor command, program instruction, and real-valued function)
IO	(monitor command, see the *V+ Operating System Reference Guide)*
RESET	(monitor command)
RUNSIG	(program instruction)
SIG.INS	(real-valued function)
SIGNAL	(monitor command and program instruction)

Syntax

```
SIG.INS (signal_num)
```

Function

Return an indication of whether or not a digital I/O signal is installed in the system, or whether or not a software signal is available in the system.

Parameter

signal_num Real-valued expression that defines the number of the digital I/O or software signal to check. (The absolute value is used, so negative signal numbers are allowed.)

Details

This function returns TRUE (–1) if the specified digital I/O or software signal is available for use by the system. Otherwise, FALSE (0.0) is returned. The function always returns TRUE if signal_number is zero.

This function can be used to make sure the digital I/O signals are installed as expected by the application program.

Example

The following program segment checks if digital I/O signal #12 is installed as an input signal (referenced as signal #1012). A message is displayed on the system terminal if the signal is not configured correctly:

```
in.sig = 1012
IF NOT SIG.INS(in.sig) THEN
   TYPE "Digital I/O signal ", in.sig, "is not installed"
END
```

Related Keywords

BITS	(monitor command, program instruction, and real-valued function)
IO	(monitor command, see the *V+ Operating System Reference Guide)*
RESET	(monitor command)
RUNSIG	(program instruction)
SIG.INS	(real-valued function)
SIGNAL	(monitor command and program instruction)

Syntax

SIGN (value)

Function

Return the value 1, with the sign of the value parameter.

Parameter

value Real-valued expression.

Details

This function returns –1.0 if the value of the parameter is less than zero. If the parameter value is greater than or equal to zero, +1.0 is returned.

Example

```
SIGN(0)         ;Returns 1.0
SIGN(0.123)     ;Returns 1.0
SIGN(–5.462)    ;Returns –1.0
SIGN(1.3125E+2);Returns 1.0
```

Syntax

```
SIGNAL signal_num, ..., signal_num
```

Function

Turn on or off external digital output signals or internal software signals.

Parameter

signal_num Real-valued expression that evaluates to a digital output or internal signal number. A positive value indicates turn on; a negative value indicates turn off. (SIGNAL ignores parameters with a zero value.)

Details

The magnitude of a **signal_num** parameter determines which digital or internal signal is to be considered. Only digital output signals (numbered from 1 to 8 and 33 to 512) and internal (software) signals (numbered from 2001 to 2512) can be specified. Only digital signals that are actually installed and configured as outputs can be used. To check your current digital I/O configuration, use the IO monitor command.

Note that software signal 2032 (brake solenoid) is a read-only signal. Attempting to set this signal will result in a *Illegal digital signal* error.

If the sign of the **signal_num** parameter is positive, the signal is turned on. If the sign of the **signal_num** parameter is negative, the signal is turned off.

Examples

Turn off the external output signal specified by the value of the variable reset (assuming the value of reset is positive), and turn on external output signal #4:

```
SIGNAL -reset, 4
```

Turn external output signal #1 off, external output signal #4 on, and internal software signal #2010 on:

```
SIGNAL -1, 4, 2010
```

Related Keywords

BITS	(monitor command, program instruction, and real-valued function)
IO	(monitor command, see the *V+ Operating System Reference Guide)*
RESET	(monitor command)
RUNSIG	(program instruction)
SIG	(real-valued function)
SIG.INS	(real-valued function)
SIGNAL	(monitor command, see the *V+ Operating System Reference Guide)*

Syntax

SIN (value)

Function

Return the trigonometric sine of a given angle.

Usage Considerations

The angle parameter must be measured in degrees.

The parameter will be interpreted as modulo 360 degrees, but excessively large values may cause a loss of accuracy in the returned value.

Parameter

value	Real-valued expression that defines the angular value to be considered.

Details

Returns the trigonometric sine of the argument, which is assumed to have units of degrees. The resulting value will always be in the range of –1.0 to +1.0, inclusive.

Examples

```
SIN(0.123)    ;Returns 2.146699E-03
SIN(-5.462)   ;Returns -0.09518546
SIN(30)       ;Returns 0.4999999
```

NOTE: TYPE, PROMPT, and similar instructions output the results of the above examples as single-precision values. However, they are actually stored and manipulated as double-precision values. The LISTR monitor command will display real values to full precision.

Syntax

SINGLE ALWAYS

Function

Limit rotations of the robot wrist joint to the range –180 degrees to +180 degrees.

Usage Considerations

Only the next robot motion will be affected if the ALWAYS parameter is not specified.

MULTIPLE ALWAYS is assumed whenever program execution is initiated and when a new execution cycle begins.

The SINGLE instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the SINGLE instruction will cause an error.

Parameter

ALWAYS	Optional qualifier that establishes SINGLE as the default condition. That is, if ALWAYS is specified, SINGLE will remain in effect continuously until disabled by a MULTIPLE instruction. If ALWAYS is not specified, the SINGLE instruction will apply only to the next robot motion.

Details

When moving to a transformation-specified location, the robot normally moves the wrist joint the minimum distance necessary to achieve the required orientation. In some cases, this action can move the wrist close to a limit stop so that a subsequent straight-line motion will hit the stop.

Specifying SINGLE will force the wrist back to near the center of its range so that straight-line motions will not fail in this way.

SINGLE is commonly specified during an APPRO to pick up an object whose position and orientation were unknown at robot programming time. Once the object is acquired, the wrist motion can be kept to a minimum.

Related Keywords

CONFIG	(real-valued function)
MULTIPLE	(program instruction)
SELECT	(program instruction and real-valued function)

Syntax

```
SOLVE.ANGLES o.jts[o.idx], o.flags, error = trans,
i.jts[i.idx],i.flags
```

Function

Compute the robot joint positions (for the current robot) that are equivalent to a specified transformation.

Usage Considerations

Since the computation performed by this instruction is a function of the geometry of the robot (link dimensions, number of axes, tool offsets, base offsets), robots with different geometric parameters will yield different results. In fact, since robots of the same general type may differ slightly in their dimensions, this instruction may return slightly different results when executed on two different robot systems of the same type.

The SOLVE.ANGLES instruction returns information for the robot selected by the task executing the instruction.

If the V+ system is not configured to control a robot, executing this instruction will not generate an error due to the absence of a robot. However, the information returned may not be meaningful.

Parameters

`o.jts` Real-valued array in which the computed joint angles are returned. The first specified element of the array will contain the position for joint #1, the second element will contain the value for joint #2, etc. For rotating joints, the joint positions will be in degrees. For translational joints, the joint positions will be in millimeters.

If a computed joint position is outside the working range for the joint, the limit stop closest to the initial joint position (as indicated by i.jts[ ]) will be returned.

`o.idx` Optional real value, variable, or expression (interpreted as an integer) that identifies the array element to receive the position for joint #1. If no index is specified, array element zero will contain the position for joint #1.

`o.flags` Real variable that receives a bit-flag value that indicates the configuration of the robot corresponding to the computed joint positions. The bit flags are interpreted as follows:

Bit 1 (LSB) RIGHTY (mask value = 1)

If this bit is set, the position has the robot in a right-arm configuration. Otherwise, the position has the robot in a left-arm configuration.

Bit 2 BELOW (mask value = 2)

If this bit is set, the position has the robot configured with the elbow below the line from the shoulder to the wrist. Otherwise, the robot elbow is above the shoulder-wrist line. (This bit is always 0 when a SCARA robot is in use.)

Bit 3 FLIP (mask value = 4)

If this bit is set, the position has the robot configured with the pitch axis of the wrist set to a negative angle. Otherwise, the pitch angle of the robot wrist has a positive value. (This bit is always 0 when the robot does not have a three-axis wrist, which is the case for a SCARA robot.)

`error` Real variable that receives a bit-flag value, which indicates if any joint positions were computed to be outside of their working range, or if the XYZ position of the destination was outside the working envelope of the robot. The bit flags are interpreted as follows:

Bits 1 - 12 If set, the computed value for the joint or motor was found to be outside of its limit stops:

Bit	Joint/Motor #	Mask value
1	1	^H1
2	2	^H2
3	3	^H4
4	4	^H8
5	5	^H10
6	6	^H20
7	7	^H40
8	8	^H80
9	9	^H100

Bit	Joint/Motor #	Mask value
10	10	^H200
11	11	^H400
12	12	^H800

Bit 14 — Too close (mask value = ^H2000)

The XYZ position of the destination could not be reached because it was too close to the column of the robot.

Bit 15 — Too far (mask value = ^H4000)

The XYZ position of the destination could not be reached because it was too far away from the robot.

Bit 16 — Joint vs. motor(mask value = ^H8000)

If set, a motor is limiting. Otherwise, a joint is limiting.

`trans` — Transformation variable, function, or compound transformation that defines the robot location of interest.

`i.jts` — Real array that contains the joint positions representing the starting location for the robot. These values are referenced: (1) for multiple-turn joints to minimize joint rotations, and (2) when a computed joint position is out of range to determine which limit stop to return.

The first specified element of the array must contain the position for joint #1. The second element must contain the value for joint #2, etc. For rotating joints, the joint positions are assumed to be in degrees. For translational joints, the joint positions are assumed to be in millimeters.

`i.idx` — Optional real value, variable, or expression (interpreted as an integer) that identifies the array element that contains the position value for joint #1. If no index is specified, element zero must contain the position for joint #1.

`i.flags` — Real value, variable, or expression whose value is interpreted as bit flags that indicate: (1) the initial configuration of the robot, (2) any changes in configuration that are to be made, and (3) special operating modes. The bit flags are interpreted as follows:

Bit 1 (LSB) RIGHTY (mask value = 1)

If this bit is set, the robot is assumed initially to be in a right-arm configuration. Otherwise, the robot is assumed to be in a left-arm configuration.

Bit 2 BELOW (mask value = ^H2)

If this bit is set, the robot is assumed initially to have its elbow below the line from the shoulder to the wrist. Otherwise, the robot is assumed to have its elbow above that line. (This bit is ignored for robots, like the SCARA configurations, that do not have an elbow that moves in a vertical plane.)

Bit 3 FLIP (mask value = ^H4)

If this bit is set, the robot is assumed initially to have the pitch axis of the wrist set to a negative value. Otherwise, the pitch angle is assumed to be set to a positive value. This bit is ignored if the robot does not have a three-axis wrist.

Bit 9 Change RIGHTY/LEFTY (mask value = ^H100)

If this bit is set, the instruction will attempt to compute a set of joint positions corresponding to the RIGHTY/LEFTY configuration specified by bit 10.

Bit 10 Change to RIGHTY (mask value = ^H200)

When bit 9 is set and this bit is set, the instruction will attempt to compute joint positions for a right-arm configuration. If bit 9 is set and this bit is 0, the instruction will attempt to compute a set of joint positions for a left-arm configuration.

Bit 11 Change BELOW/ABOVE (mask value = ^H400)

If this bit is set, the instruction will attempt to compute a set of joint positions corresponding to the BELOW/ABOVE configuration specified by bit 12. This bit is ignored for robots, like the SCARA configurations, that do not have an elbow that moves in a vertical plane.

Bit 12 Change to BELOW (mask value = ^H800)

When bit 11 is set and this bit is set, the instruction will attempt to compute joint positions for an elbow-down configuration. If bit 11 is set and this bit is 0, the instruction will attempt to compute joint positions for an elbow-up configuration. This bit is ignored for robots, like the SCARA configurations, that do not have an elbow that moves in a vertical plane.

Bit 13 Change FLIP/NOFLIP (mask value = ^H1000)

If this bit is set, the instruction will attempt to compute a set of joint positions corresponding to the FLIP/NOFLIP configuration specified by bit 14. This bit is ignored if the robot does not have a three-axis wrist.

Bit 14 Change to FLIP (mask value = ^H2000)

When bit 13 is set and this bit is set, the instruction will attempt to compute joint positions for a FLIP wrist configuration. If bit 13 is set and this bit is 0, the instruction will attempt to compute joint positions for a NOFLIP wrist configuration. This bit is ignored if the robot does not have a three-axis wrist.

Bit 21 Avoid degeneracy (mask value = ^H100000)

When this bit is set, if the computed value of joint #2 is within 10 degrees of having the outer link straight out (that is, joint 2 between –10 and +10 degrees in value), an out-of-range error for joint 2 will be signaled.

Bit 22 Single-turn joint 4 (mask value = ^H200000)

When this bit is set, the computed value of joint 4 will be restricted to the range of –180 to +180 degrees.

Bit 23 Straight-line motion (mask value = ^H400000)

When this bit is set, the joint positions returned must correspond to the same configuration as those initially specified. That is, no change in robot configuration is allowed.

Details

This instruction computes the joint positions that are equivalent to a specified transformation value using the geometric data of the robot connected to the system. The specified transformation is interpreted to be the position and location of the end of the robot tool in the World coordinate system, taking into consideration the current TOOL transformation and BASE offsets.

Example

The instructions below do not perform any useful function but are intended to illustrate how the SOLVE.ANGLES instruction operates. After execution of these instructions, both the jts2 and jts arrays will contain approximately the same values. Any differences in the values are due to computational round-off errors:

```
HERE #cpos
DECOMPOSE jts[] = #cpos
SOLVE.TRANS new.t, error = jts[]
SOLVE.ANGLES jts2[], flags, error = new.t, jts[],
SOLVE.FLAGS(jts[])
```

Related Keywords

DECOMPOSE (program instruction)

SELECT (program instruction and real-valued function)

SOLVE.FLAGS (real-valued function)

SOLVE.TRANS (program instruction)

Syntax

SOLVE.FLAGS (joints[index**])**

Function

Return bit flags representing the robot configuration specified by an array of joint positions.

Usage Considerations

The SOLVE.FLAG function returns information for the robot selected by the task executing the function.

If the V+ system is not configured to control a robot, use of the SOLVE.FLAGS function will cause an error.

Parameters

joints	Real array that contains the robot joint positions. The first specified element of the array must contain the position for joint #1, the second element must contain the value for joint #2, etc. For rotating joints, the joint positions are assumed to have units of degrees. For translational joints, the joint positions are assumed to have units of millimeters.
index	Optional real value, variable, or expression (interpreted as an integer) that identifies the array element that contains the position for joint #1. If no index is specified, element zero must contain the position for joint #1.

Details

This function returns bit flags that indicate the configuration of the robot (for example, RIGHTY or LEFTY) for a given set of joint positions. This function is useful for providing the configuration data required by the SOLVE.ANGLES program instruction.

The bits of the value returned by this function are interpreted as follows:

Bit 1 (LSB) RIGHTY (mask value = 1)

If this bit is set, the position has the robot in a right-arm configuration. Otherwise, the position is for a left-arm configuration.

Bit 2 BELOW (mask value = 2)

If this bit is set, the position has the robot configured with the elbow below the line from the shoulder to the wrist. Otherwise, the robot elbow is above the shoulder-wrist line. (This bit is always 0 when a SCARA robot is in use.)

Bit 3 FLIP (mask value = 4)

If this bit is set, the position has the robot configured with the pitch axis of the wrist set to a negative angle. Otherwise, the wrist pitch angle has a positive value. (This bit is always 0 when the robot does not have a three-axis wrist, as is the case for a four- axis SCARA robot.)

Related Keywords

ABOVE/**BELOW**	(program instructions)
DECOMPOSE	(program instruction)
LEFTY/**RIGHTY**	(program instructions)
FLIP/**NOFLIP**	(program instructions)
SELECT	(program instruction and real-valued function)
SOLVE.ANGLES	(program instruction)
SOLVE.TRANS	(program instruction)

Syntax

```
SOLVE.TRANS transform, error = joints[index]
```

Function

Compute the transformation equivalent to a given set of joint positions for the current robot.

Usage Considerations

Since the computation performed by this instruction is a function of the geometry of the robot (link dimensions, number of axes, tool offsets, base offsets), robots with different geometric parameters will yield different results. In fact, since robots of the same general type may differ slightly in their dimensions, this instruction may return slightly different results when executed on two different robot systems of the same type.

The SOLVE.TRANS instruction refers to the robot selected by the task executing the instruction.

If the V+ system is not configured to control a robot, executing the SOLVE.TRANS instruction will not generate an error due to the absence of a robot. However, the information returned may not be meaningful.

Parameters

transform	Transformation variable or transformation array element in which the result is stored.
error	Real variable that is set to a V+ error code if a computational error occurred during processing of the instruction. This variable is set to 0 if no error occurs. (The only error that is currently reported is arithmetic overflow [–409], so this parameter can be considered as returning a TRUE or FALSE value.)
joints	Real-valued array that contains the joint positions that are to be converted to an equivalent transformation. The first specified element of the array must contain the position for joint #1, the second element must contain the value for joint #2, etc. For rotating joints, the joint positions are assumed to have units of degrees. For translational joints, the joint positions are assumed to have units of millimeters.
index	Optional integer value that identifies the array element that contains the position for joint #1. If no index is specified, element zero must contain the position for joint #1.

Details

This instruction converts a set of joint positions to an equivalent transformation value using the geometric data of the robot connected to the system. The computed transformation represents the position and orientation of the end of the tool in the World coordinate system taking into consideration the current TOOL transformation and BASE offsets.

Example

The series of instructions below computes the position and orientation that the robot will be moved to if its current location is altered by rotating joint #1 by 10 degrees:

```
HERE #cpos
DECOMPOSE joints[1] = #cpos
joints[1] = joints[1]+10
SOLVE.TRANS new.trans, error = joints[1]
```

Related Keywords

DECOMPOSE (program instruction)

SELECT (program instruction and real-valued function)

SOLVE.ANGLES (program instruction)

SOLVE.FLAGS (real-valued function)

Syntax

```
SPEED speed_factor, r_speed_factor units ALWAYS
```

Function

Set the nominal speed for subsequent robot motions.

Usage Considerations

SPEED 100,100 ALWAYS is assumed whenever program execution is started and when a new execution cycle begins.

Motion speed has different meanings for joint-interpolated motions and straight-line motions.

The speed of robot motions is determined by a *combination* of the program speed setting and the monitor speed setting.

The SPEED instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the SPEED instruction causes an error.

Parameters

speed_factor	Real value, variable, or expression whose value is used as a new speed factor. The value 100 is considered normal full speed, 50 is 1/2 of full speed, and so on. If IPS or MMSPS is specified for `units`, the value is considered the linear tool tip speed.
r_speed_factor	Optional real value, variable, or expression whose value is used as a new straight-line motion rotational speed factor. The value 100 is considered normal full speed, 50 is 1/2 of full speed, etc.
units	Optional keyword—either IPS (for inches per second), MMPS (for millimeters per second), or MONITOR—that determines how to interpret the **speed_factor** parameter.
ALWAYS	Optional keyword. If specified, the program **speed_factor** will be in effect until the next SPEED instruction changes program speed. Otherwise, it will be in effect only for the next motion instruction (including APPROaches and DEPARTs).

Details

If the `units` parameter is omitted, this instruction determines the program speed—the nominal robot motion speed assuming that the monitor speed factor is 100%.

If MONITOR is specified for `units`, the monitor speed is set. In this case, the parameter `r_speed_factor` is ignored and `ALWAYS` is assumed. The speed at which motions are actually performed is determined by combining the values specified in this instruction with the current program speed setting. Monitor speed changes take place immediately, including the remaining portion of a currently executing move.

If IPS or MMPS is specified in the `units` parameter, **`speed_factor`** is interpreted as the absolute tool-tip speed for *straight-line* motions. In this case, the **`speed_factor`** parameter has no direct meaning for joint-interpolated motions.

The effects of changing program speed and monitor speed differ slightly for continuous-path motions. As the robot moves through a series of points, the robot comes as close to the points as possible while maintaining the program speed and specified accelerations. As program speed increases, the robot will make coarser approximations to the actual point in order to maintain the program speed and accelerations.

When the monitor speed is increased, the path of the robot relative to the commanded destination points is not altered but the accelerations will be increased. For applications where path following is important, the path can be defined with the monitor speed set to a low value, and then accurately replayed at a higher monitor speed.

Speed cannot be less than 0.000001 (1.0E–6).

During straight-line motions, if a tool with a large offset is attached to the robot, the robot joint and flange speeds could be very large when rotations about the tool tip are made. The `r_speed_factor` parameter permits control of the maximum tool rotation speeds during straight-line motions.

If a rotational speed factor (`r_speed_factor`) is specified, it is interpreted as a percentage of maximum Cartesian rotation speed to be used during straight-line motions. If the `r_speed_factor` parameter is not specified, one of the following results occurs:

1. If the `units` parameter is also omitted, the rotational speed is set to the value of **`speed_factor`**.
2. If the `units` parameter is specified, the rotational speed is not changed.

When IPS or MMPS are specified, the **speed_factor** is converted internally to the corresponding nominal speed. If the SPEED real-valued function is then used to read the program speed, the value returned will be a percentage speed factor and not an absolute speed setting.

Remember, the final robot speed is a combination of the monitor speed (SPEED monitor command), the program speed (SPEED instruction), and the acceleration or deceleration (ACCEL program instruction).

Examples

Set the program speed to 50% for the next motion (assuming the monitor speed is 100):

```
SPEED 50
```

Set the nominal tool tip speed to 20 inches per second (assuming the monitor speed is 100) for straight-line motions. Rotations about the tool tip will be limited to 40% of maximum. The settings will remain in effect until changed by another SPEED instruction.

```
SPEED 20, 40 IPS ALWAYS
```

Set the monitor speed to 50% of normal:

```
SPEED 50 MONITOR
```

Related Keywords

ACCEL	(program instruction)
DURATION	(program instruction)
IPS	(conversion factor)
MMPS	(conversion factor)
PAYLOAD	(program instruction)
SCALE.ACCEL	(system switch)
SELECT	(program instruction and real-valued function)
SPEED	(monitor command and real-valued function)

Syntax

```
SPEED (select)
```

Function

Return one of the system motion speed factors.

Usage Considerations

The SPEED function returns information for the robot selected by the task executing the function.

If the V+ system is not configured to control a robot, use of the SPEED function will not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

Parameter

`select` Real-valued expression whose value determines which speed factor should be returned (see below).

Details

This function returns the system motion speed factor corresponding to the select parameter value. The acceptable parameter values, and the corresponding speed values returned, are:

select	Speed value returned
1	Monitor speed (set by SPEED monitor command)
2	Permanent program speed (set by a SPEED ... ALWAYS program instruction)
3	Temporary program speed for the last or current motion
4	Temporary program speed to be used for the next motion
5	Permanent program rotation speed
6	Temporary program rotation speed for the last or current straight-line motion
7	Temporary program rotation speed to be used for the next straight-line motion
8	The maximum allowable setting for program speed

Note that the value returned should be interpreted as a percentage of normal speed, even if the program speed was set by a SPEED program instruction that specified a speed setting. (See the SPEED program instruction.)

Example

The following program segment makes one motion at 1/2 of the permanent program speed:

```
new.speed = SPEED(2)/2  ;Compute 1/2 the permanent speed
SPEED new.speed         ;Move at the new speed next time
MOVE pick.up            ;Perform the actual motion
```

Note that the following instruction sequence is equivalent:

```
SPEED SPEED(2)/2        ;Reduce speed for the next motion
MOVE pick.up            ;Perform the actual motion
```

Related Keywords

ACCEL (real-valued function)

DURATION (real-valued function)

SELECT (program instruction and real-valued function)

SPEED (monitor command and program instruction)

Syntax

```
SPIN speeds[index]
```

Function

Rotate one or more joints of the selected robot at a specified speed.

Usage Considerations

This instruction currently can be used only with the Joints (JTS) robot device module.

The acceleration and deceleration factors used for SPIN can be changed in the normal way, using the V+ ACCEL instruction. Because SPIN uses a constant acceleration rate, the optional profile parameter of the ACCEL instruction is ignored by the SPIN instruction. V+ limits the acceleration rate and deceleration rate to be less than or equal to the maximum values that are defined for the robot. (These maximum values are specified by the system designer using the SPEC utility program.)

The SPEED program instruction does not affect SPIN. However, the SPEED monitor command does modify the SPIN speed.

Parameters

`speeds[]` The optional `speeds[]` array contains the rate at which each joint of the selected robot is to be rotated. The rates are in units of degrees/second or millimeters/second. Joints can be rotated in either the positive or negative direction. V+ limits the spin speeds to be less than or equal to the 100% joint-speed values that are defined for the robot.

Other facts about the `speeds[]` array:

- If the array is omitted, all values default to zero.
- If the array is specified, all the expected elements must be defined. Sparse arrays generate an *Undefined value* error message.
- For all array types (AUTO, etc.), the highest element index must be equal to or greater than what is required.

`index` Optional integer value that identifies which array element is applied to joint1. Zero is assumed if omitted. Joint 2 uses the value from element [`index`+1], and so on.

CAUTION: Take care to set the index value correctly, otherwise unexpected joint speeds may result.

Details

The SPIN instruction uses a constant acceleration rate to accelerate each joint up to its specified speed. Once each joint has reached its specified speed, it will continue to rotate at a constant speed until one of the following occurs:

- Another SPIN instruction is executed that changes one or more of the SPIN rates.
- A SPIN instruction is executed that decelerates all of the joints to a stop.
- An error occurs (PANIC, BRAKE, joint limit stop encountered, etc.) that decelerates all joints to a stop.

To use the SPIN instruction to stop all of the joints, the SPIN instruction must be executed with either no `speeds[]` array specified or with all of the array elements set to 0.

If a SPIN operation has been abnormally terminated by a PANIC stop or other error condition, the spin operation can be resumed by issuing a RETRY monitor command.

SPIN movements cannot be blended with the standard motion instructions (such as MOVE, MOVES, APPRO, DEPART). Therefore, a BREAK is implicitly executed before a SPIN instruction to allow any standard motion to complete. (If there is more than one SPIN instruction in a sequence, then an implicit BREAK is executed only before the first SPIN instruction.)

Also, SPIN trajectories must be stopped with a SPIN instruction, BRAKE, etc., prior to executing a regular motion instruction. If a SPIN trajectory is being executed, and a regular motion instruction is executed, the following error message is generated:

```
*Illegal while joints SPIN'ing*
```

The SPIN instruction can only be used to move a joint that has been configured with the continuous-rotation capability. If a SPIN instruction attempts to move a joint that has not been so configured, or if the robot is currently tracking a belt or moving under ALTER control, the following error message is generated:

```
*SPIN motion not permitted*
```

If any joints of the selected robot cannot rotate continuously, their corresponding spin rates should always be specified as 0.

When a SPIN instruction is executed, the value of the DEST function is not meaningful and the value of the #PDEST function returns the specified joint speeds instead of the final joint angles.

Continous-Turn Axes

In order for a joint to rotate continuously, the following conditions must be satisfied: (1) the robot device module must be designed to support one or more continuous-turn joints, (2) the encoder roll-over value for the joint must be nonzero, and (3) the joint stop limits must be set to values substantially greater than the roll-over value.

Currently, only the joints (JTS) robot device module has been designed to support continuous-turn joints, and any of its joints may be configured for continuous turning. To set the roll-over value for a joint, the SPEC utility program must be run.

V+ uses the roll-over value in the following manner: At selected times V+ compares the absolute value of the commanded position of the joint to its roll-over value and, if the commanded position exceeds the roll-over value, the roll-over value is subtracted from (or added to) the commanded position. This test for joint roll-over is performed at the following times: (1) at the end of a continuous-path motion; (2) whenever the robot is in FREE/JOINT/WORLD/TOOL manual control modes; (3) whenever robot power is disabled; and (4) whenever a joint is being moved via a SPIN instruction.

Example

Assign spin speeds for all joints of the currently selected robot which is assumed to have four axes:

```
temp.speeds[1] = 0.5    ;Turn J1 at 30 rpm
temp.speeds[2] = 0.25   ;Turn J2 at 15 rpm
temp.speeds[3] = 0      ;Do not turn J3
temp.speeds[4] = 1.0    ;Turn J4 at 1 rps (30 rpm)
SPIN temp.speeds[1]     ;Start the motion
```

Syntax

SQR (value)

Function

Return the square of the parameter.

Parameter

value Real-valued expression whose value is to be squared.

Details

This is a convenience function that computes the square of a value. That is, the result is equal to (value * value).

Examples

```
SQR(0.123)       ;Returns 0.015129
SQR(4)           ;Returns 16
SQR(-5.462)      ;Returns 29.83344
SQR(1.3125E+2)   ;Returns 17226.56
```

NOTE: TYPE, PROMPT, and similar instructions output the results of the above examples as single-precision values. However, they are actually stored and manipulated as double-precision values. The LISTR monitor command will display real values to full precision.

Syntax

SQRT (value)

Function

Return the square root of the parameter.

Parameter

value Real-valued expression defining the value whose square root is to be computed.

Details

Returns the square root of the argument if the argument is greater than zero. An error results if the argument is less than zero.

The square root of a number is defined to be the number that, when multiplied by itself, yields the original number.

Examples

```
SQRT(0.123)      ;Returns 0.3507136
SQRT(4)          ;Returns 2.0
SQRT(-5.462)     ;Returns *Negative square root*
SQRT(1.3125E+2)  ;Returns 11.45644
```

NOTE: TYPE, PROMPT, and similar instructions output the results of the above examples as single-precision values. However, they are actually stored and manipulated as double-precision values. The LISTR monitor command will display real values to full precision.

Syntax

```
STATE (select)
```

Function

Return a value that provides information about the robot system state.

Usage Considerations

The STATE function returns information for the robot selected by the task executing the function.

Parameter

`select`	Real value, variable, or expression (interpreted as an integer) that selects the category of state information returned, as described below.

Details

When **`select`** = 1, the function value returns information about the overall robot state as follows:

Value	**Interpretation** (when **`select`** = 1)
0	Resetting system after robot power has been turned off.
1	A fatal error has occurred and robot power cannot be turned on.
2	Waiting for user to turn on robot power.
3	Robot power was just turned on; initialization is occurring.
4	Manual control mode is active.
5	A CALIBRATE command or instruction is executing.
6	Not used.
7	Robot is under program control.
8	Robot power is on; robot is not calibrated and cannot be moved.

When **`select`** = 2, the function value returns information about the current or previous robot motion. These modes can change only when the robot is under program control—that is, when STATE(1) = 7:

Value	Interpretation (when **select** = 2)
0	No motion instructions executed yet.
1	Normal trajectory evaluation is in progress (including normal acceleration, deceleration and segment transitions).
2	Motion stopped at a planned location.[a]
3	Position error is being nulled at unplanned final location.
4	Motion stopped at an unplanned location due to a belt window violation.[b]
5	Decelerating due to a triggered REACTI or BRAKE instruction.
6	Stopped due to a triggered REACTI or BRAKE instruction.[b]
7	Decelerating due to a hardware error, panic button, or ESTOP instruction.
8	Stopped due to a hardware error, panic button, or ESTOP instruction.[b]
9	Decelerating due to a stop-on-force condition.
10	Stopped due to a stop-on-force condition.
11	Nulling at completion of a SPIN motion.
12	Stopped after completion of a SPIN instruction.

[a] A RETRY command has no effect.
[b] A RETRY command completes the previous motion

When **select** = 3, the function value returns information about the current manual control mode as follows:

Value	Interpretation (**select** = 3)
0	Manual mode without selection.
1	Free-joint mode.
2	Individual joint control.
3	World coordinates control.
4	Tool coordinates control.
5	Computer control enabled.

When **select** = 4, the function value returns information about the controller's front-panel control settings and other hardware status to be read by programs. Interpret the value as a set of bit flags, each of which indicates a corresponding condition. STATE(4) always returns with bit 1 set.

Bit mask	Interpretation when bit set (**select** = 4)
1	Front panel hardware is connected.
2	The PROGRAM START button is pushed.
4	A hardware panic button is pushed.
8	The HIGH POWER ON/OFF button is pushed.

When **select** = 5, the function value indicates the settings of the controller keyswitches on the external front panel (VFP). (For information on this panel, refer to the ***Adept MV Controller User's Guide***.)

Value	Interpretation (**select** = 5)
0	The VFP is disconnected.
1	Both AUTO and LOCAL are set.
2	Both MANUAL and LOCAL are set.
3	Both AUTO and REMOTE are set.
4	Both MANUAL and REMOTE are set.

When **select** = 6, the function returns an indication of whether or not the real-time path-modification facility (alter mode) is enabled. If zero is returned, alter mode is disabled for the current motion. If a nonzero value is returned, alter is enabled, and the low byte of this value contains bits that correspond to the mode specified in the ALTON instruction that initiated the path modification.

When **select** = 7, the function returns an indication of whether or not the real-time path-modification facility (alter mode) is enabled for the next planned motion. If zero is returned, alter mode is disabled for the next motion. If a nonzero value is returned, alter is enabled, and the low byte of this value contains bits that correspond to the mode specified in the ALTON instruction that initiated the path modification. (This option is available only if your Adept system is equipped with the V+ Extensions option.)

When **select** = 8, the number of the robot selected by the manual control pendant is returned.

When **select** = 9, the function returns the time (in seconds) left until completion of the current motion. Zero indicates that no motion is in progress. For continuous-path motions, the value of STATE(9) decreases during each motion until the transition to the next motion, and then the value suddenly changes to the time left in the next motion. That is, STATE(9) does not reach 0 before it is reset to reflect the next motion.

When **select** = 10, the function returns the percentage of the current motion that has completed. The value 100 indicates that no motion is in progress. For continuous-path motions, the value of STATE(10) increases during each motion until the transition to the next motion, and then the value suddenly changes to close to 0 to reflect the start of the next motion. That is, STATE(10) does not reach 100 before it is reset to reflect the next motion.

When **select** = 11, the function returns detailed information on which portion of the acceleration profile is currently being generated for the selected robot.

Value	**Interpretation** (**select** = 11)
0	Idle, not evaluating trajectory
1	Ramping up acceleration for new segment
2	Constant acceleration section
3	Ramping down acceleration
4	Constant velocity section
5	Ramping up acceleration for the next motion during the transition section between motions
6	Constant acceleration for the next motion during the transition section between motions
7	Ramping down acceleration for the next motion during the transition section between motions
8	Ramping up deceleration
9	Constant deceleration
10	Ramping down deceleration
11	Nulling final errors

When **select** = 12, the function returns a flag that is set to nonzero when an ALTER program instruction is executed for the currently selected robot, and cleared after the trajectory generator processes the posted ALTER data. This flag can be used to coordinate the execution of ALTER instructions with the processing of the data by the trajectory generator.

When **select** = 13, the function returns the trajectory generator execution rate in hertz. That is, if the trajectory generator is executed once each major V+ cycle, this function returns the value 62.5.

When **select** = 14, the function returns the servo code execution rate in hertz. That is, if the servos are executed each 1 msec, this function returns value 1000 (1kHz).

When **select** = 15, the function returns the number of the motion that is being executed by the selected robot. This number is zeroed when a program that is attached to the robot first begins executing. The counter is reset to 1 at the start of each EXECUTE cycle. The value is incremented each time the trajectory generator begins evaluating a new motion (or transitions to a new continuous-path motion). The value of STATE(15) ranges from 0 to ^HFFFF. After reaching ^HFFFF, the value rolls back to 0.

Example

The following example shows how the STATE function can be used to determine whether or not a REACTI was triggered during a robot motion:

```
REACTI 1001       ;Setup the reaction
MOVES final       ;Start the robot motion
BREAK             ;Wait for the motion to complete

CASE STATE(2) OF ;Decide what happened

  VALUE 2:
  TYPE "Motion completed normally"

  VALUE 6:
  TYPE "Motion stopped by REACTI"

  VALUE 8:
  TYPE "Motion stopped by panic button"

END
```

Related Keywords

CONFIG	(real-valued function)
NETWORK	(real-valued function)
SELECT	(program instruction and real-valued function)
STATUS	(monitor command and real-valued function)
TASK	(real-valued function)

Syntax

```
STATUS (program_name)
```

Function

Return status information for an application program.

Parameter

`program_name` String constant, variable, or expression that specifies the name of the application program of interest. Letters in the name can be uppercase or lowercase. The string can be empty () in order not to specify a program name (see below), but the parameter cannot be omitted.

Details

This function returns information about the execution status of the specified program.

If no program name is specified (that is, the parameter string is empty []), the task number of the program containing the function call is returned. This allows a program to determine which system task it is executing as. (Tasks and task numbers are described in the ***V+ Language User's Guide***.)

If a program name is specified as the function parameter, the status of that program is returned as follows:

Value returned	Program status
–1	Not executing.
–2	Not defined.
–3	Interlocked because of write.
–4	Not executable.
–5	Interlocked because of read.

A program is considered write-interlocked when it is being copied, deleted, renamed, or edited in read-write mode. A program is considered read-interlocked when it is executing (by one or more tasks) or is being edited in read-only mode.

A program is considered not executable when it contains a structure error or a bad line.

NOTE: If a program is being executed by multiple tasks, the STATUS function will return –5. There is no way to use the STATUS function to determine when the program ceases to be executed by one of those tasks. The STATUS function will not return –1 until *all* the tasks stop executing the program.

The function will return not defined status if an invalid program name is specified (for example, if the name does not start with a letter).

Example

The following program segment demonstrates how the STATUS function can be used to decide whether or not to initiate execution of an application program:

```
IF STATUS("pc.main") == -1 THEN
   EXECUTE 1 pc.main
END
```

NOTE: The STATUS function will not return –1 if the program is being executed by *any* program task. Thus, this example may not be appropriate for some situations. (See the example shown for the EXECUTE instruction for another technique for initiating execution of another program task.)

Related Keywords

DEFINED	(real-valued function)
STATE	(real-valued function)
STATUS	(monitor command)
TASK	(real-valued function)
TESTP	(monitor command, see the ***V+ Operating System Reference Guide)***

Syntax

STOP

Function

Terminate execution of the current program cycle.

Usage Considerations

STOP will not halt program execution if there are more program cycles to execute.

The PROCEED command cannot be used to resume program execution after a STOP instruction causes the program to halt.

If program execution is halted by a STOP instruction, FCLOSE and/or DETACH are forced on the disk and serial communication logical units as required.

Details

Counts one more program cycle as complete and one less remaining. If the result is that no more cycles are remaining, program execution halts.

If more cycles are remaining, the internal robot motion parameters are reinitialized, and program execution continues with the first step of the main program (even if the STOP occurred within a subroutine or reaction program).

Terminates execution of the current program unless more program loops (see the EXECUTE command and instruction) are to be completed, in which case execution of the program continues at its first step. Thus, the STOP instruction is used to mark the end of a program execution pass. Note that the HALT instruction has a different effect—it cancels all remaining cycles.

A RETURN instruction in a main program has the same effect as a STOP instruction. A main program is one that is invoked by an EXECUTE command or instruction, or a PRIME or XSTEP command, whereas a subroutine is a program that is invoked by a CALL or CALLS instruction (or a reaction) within another program.

Related Keywords

ABORT	(monitor command and program instruction)
HALT	(monitor command and program instruction)
PAUSE	(program instruction)
RETURN	(program instruction)

Syntax

```
STRDIF ($a, $b)
```

Function

Compare two strings byte by byte for the purpose of sorting.

Parameters

$a	A string constant, variable, or expression that contains the bytes to be compared with those in **$b**.
$b	A string constant, variable, or expression that contains the bytes to be compared with those in **$a**.

Details

This function compares strings byte by byte, using the unsigned byte values without any case conversion. That is, the function ignores the setting of the UPPER system switch. The two strings can have different lengths. The returned values and their meanings are as follows:

Returned value	Interpretation
–1	**$a** is less than **$b.**
0	**$a** is exactly the same as **$b**.
1	**$a** is greater than **$b.**

Note that the value is FALSE (0) if the strings are the same.

Example

Sort two names in alphabetical order:

```
$name[0] = "Michael"
$name[1] = "MARK"
CASE STRDIF($name[0],$name[1]) OF
   VALUE –1,0:
      $list[0] = $name[0]
      $list[1] = $name[1]
   VALUE 1:
        $list[0] = $name[1]
        $list[1] = $name[0]
   END
TYPE "Names in alphabetic order:  ", $list[0], "  ",
$list[1]
```

Syntax

```
SWITCH switch_name = value

SWITCH  switch_name[index] = value
```

Function

Enable or disable a system switch based on a value.

Usage Considerations

If the specified switch accepts an index qualifier and the index is zero or omitted (with or without the brackets), **all** the elements of the switch array are set according to the value given.

Parameters

switch_name	Name of the switch whose setting is to be modified. The switch name can be abbreviated to the minimum length that identifies it uniquely.
index	For switches that can be qualified by an index, this is an optional real value, variable, or expression that specifies the specific switch element of interest (see above).
value	Real value, variable, or expression that determines if the switch is to be enabled or disabled. The switch is enabled if the value is TRUE (nonzero). The switch is disabled if the value is FALSE (zero).

Details

Sets the given system switch to the setting implied by the value on the right of the equal sign.

The switch name can be abbreviated to the minimum length that identifies it uniquely.

The switch names acceptable with the standard V+ system are summarized in the *V+ Language User's Guide*. Each of the switches is described in detail elsewhere in this appendix.

Other system switches are available when options are installed. Refer to the option documentation for details. For example, the switches associated with the AdeptVision options are described in the *AdeptVision Reference Guide*.

Example

The following program statements show how the SWITCH real-valued function and instruction can be used to save the setting of a system switch, and later restore it, respectively:

```
old.upper = SWITCH(UPPER);Save the current setting
   .
   .                      ;Instructions that may change the
   .                      ;setting of the UPPER switch.
SWITCH UPPER = old.upper ;Restore the initial setting
```

Related Keywords

DISABLE (monitor command and program instruction)

ENABLE (monitor command and program instruction)

SWITCH (monitor command and real-valued function)

Syntax

SWITCH (switch_name)

SWITCH (switch_name[index**])**

Function

Return an indication of the setting of a system switch.

Parameters

switch_name	Name of the system switch of interest (see below).
index	For switches that can be qualified by an index, this is a (required) real value, variable, or expression that specifies the specific switch element of interest.

Details

This function returns FALSE (0.0) if the specified switch is disabled. Otherwise, TRUE (–1) is returned.

The switch name can be abbreviated to the minimum length that identifies it uniquely.

The switch names acceptable with the standard V+ system are summarized in the ***V+ Language User's Guide***.

Other system switches are available when options are installed. Refer to the option documentation for details. For example, the switches associated with the AdeptVision options are described in the ***AdeptVision Reference Guide***.

Example

This program segment checks whether the DRY.RUN switch is enabled. If it is, a message is displayed on the system terminal:

```
IF SWITCH(DRY.RUN) THEN
   TYPE "DRY RUN mode is enabled"
END
```

Related Keywords

DISABLE (monitor command and program instruction)

ENABLE (monitor command and program instruction)

SWITCH (monitor command and program instruction)

Syntax

$SYMBOL (pointer)

Function

Determine the user symbol that is referenced by a pointer previously obtained with the SYMBOL.PTR real-valued function.

Usage Considerations

The pointer value must have been obtained with the SYMBOL.PTR real-valued function.

Parameter

pointer Real variable that identifies the symbol to be referenced.

Details

This function can be used to determine the user symbol (that is, program or variable name) that is pointed at by a pointer previously determined with the SYMBOL.PTR real-valued function.

A null string is returned if the pointer value is zero or invalid, or if the symbol has been deleted since the pointer was defined.

Example

After the SYMBOL.PTR function has been used to set the values of elements of the array **my.pgm.ptr[ ]** (for example, see the dictionary page for the CALLP instruction), the following instruction can be used to display the program name that is referenced by one of the pointers:

```
TYPE "Program", index, " is ", $SYMBOL(my.pgm.ptr[index])
```

Related Keyword

SYMBOL.PTR (real-valued function)

Syntax

SYMBOL.PTR (string, type)

Function

Determine the value of a pointer to a user symbol in V+ memory.

Usage Considerations

The value returned by the function is meaningful only to the CALLP instruction and the $SYMBOL string function.

Parameters

string	String constant, variable, or expression that defines the symbol to be referenced.
type	Optional real value, variable, or expression that specifies the type of symbol to be referenced. Currently the only value supported is zero, which specifies that the string parameter defines a program name. The value zero applies if the parameter is omitted.

Details

The SYMBOL.PTR function can be used to obtain a pointer to a user symbol (that is, a program or variable name) in V+ memory. Such a pointer can then be used elsewhere in the program by the CALLP instruction and the $SYMBOL function. Refer to the descriptions of those keywords for more information.

The function returns the value zero if the specified symbol is not defined.

Example

Refer to the dictionary page for the CALLP instruction.

Related Keywords

CALLP	(program instruction)
$SYMBOL	(string function)

Syntax

```
TAS (variable, new_value)
```

Function

Return the current value of a real-valued variable and assign it a new value. The two actions are done indivisibly so that no other program task can modify the variable at the same time.

Parameters

`variable` Name of the real-valued variable to be tested and assigned the new value given. (If the variable is not defined when the function is executed, the function returns the value 0.)

`new_value` Real value, variable, or expression that defines the new value to be assigned to the specified variable.

Details

Because the different program tasks execute simultaneously, time-sharing the system processor, it is possible for any task to be interrupted by another in the middle of performing some computation or storing data into variables. When data is shared by two or more tasks, the programs must implement an interlock scheme to prevent the data from being accessed when it is only partially updated.

The TAS function provides a means to implement such an interlock. A program can use this function to set a control variable to a value that indicates (to the other programs) that the data is being accessed. Then, after the data is stable, the program can use TAS to set the control variable to a different value to indicate the data is available.

Without the TAS function, a much more complicated polling scheme would be needed to administer the control variable (to prevent more than one program from setting the control variable simultaneously).

Example

The following example shows how the TAS real-valued function can be used to insure exclusive access by an application program to data that is also used by another program task. (A similar instruction sequence must be used in the other application program when it wants to access the data.)

The real-variable data.locked is set to FALSE when the data is not interlocked and set to TRUE when the data is interlocked. This variable is set to TRUE with the TAS function, so that we can detect if the other program task has already set it to TRUE. Since TAS tests and sets the value indivisibly, there is no chance of both programs setting data.locked to TRUE simultaneously without the conflict being detected:

```
WHILE TAS(data.locked, TRUE) DO ;Wait until "data.locked" is
                                ;FALSE and then set to TRUE
   WAIT
END
;  ... Perform desired operations with/to the data ...
data.locked = FALSE             ;Release the data structure
```

The WHILE loop causes program execution to be blocked until the variable data.locked has the value FALSE. Thus, the program will be blocked if the other program is accessing the data array (and has locked the semaphore variable).

Once the program gains exclusive access to the data array, it can safely access the data.

The last instruction releases the data for access by the application executing as the other program task.

Related Keyword

IOTAS (real-valued function)

Syntax

```
TASK (select, task_num)
```

Function

Return information about a program execution task.

Parameters

`select`	Optional real-valued expression that has a value of 0, 1, or 2 and selects the category of task information returned (see below). The value 0 is assumed if the parameter is omitted.
`task_num`	Optional integer value that specifies which system program task is to be accessed (see below).

Details

This function returns various information about the system program execution tasks. (See the *V+ Language User's Guide* for an explanation of execution tasks.)

The `select` parameter determines the type of information that will be returned as follows:

`select` = 0 **Task number**—The function returns the number of the task executing the current program.

`select` = 1 **Task run state**—Returns the run state for the task specified by the task_number parameter. The value returned should be interpreted as follows:

Value	Interpretation
–1	Invalid task number.
0	Idle.
1	Stopped due to program completion.
2	Stopped due to program execution error (for example, undefined value).
3	Stopped due to ABORT, breakpoint, panic button pressed, robot error, single-step execution, or watchpoint.
4	Executing.

select = 2 **Task status bits**—Returns an integer value that should be interpreted as a set of bit flags that indicate the following information about the task specified by the task_number parameter:

Bit #	Bit mask	Indication if bit is set
1	1	Debugger is accessing task
2	2	Task has robot attached

Examples

Display the task number the program is running in:

```
TYPE "This program is running as task number :" TASK()
```

The following program segment demonstrates how the TASK function can be used to decide whether or not to initiate execution of a program (named pc.job.2) with task #2:

```
IF TASK(1, 2) <> 4 THEN              ;If task #2 not executing
   IF STATUS("pc.job.2") == -1 THEN;and if program is okay
     EXECUTE 2 pc.job.2              ;start it up
   ELSE                              ;But if program not okay
     TYPE /B, "Can't start task #2" ;output error message
   END
END
```

Related Keywords

STATE (real-valued function)

STATUS (monitor command and real-valued function)

Syntax

```
... TERMINAL
```

Function

Determine how V+ will interact with the system terminal.

Usage Considerations

The current value of the TERMINAL parameter can be determined with the PARAMETER monitor command or real-valued function.

The value of the TERMINAL parameter can be modified only with the PARAMETER monitor command or program instruction.

The acceptable parameter values are 0 through 4. The default value is 4 on V+ system disks supplied by Adept.

The parameter name can be abbreviated.

Details

The possible TERMINAL parameter values are described below:

Parameter value	Terminal type	Treatment of DEL & BS	Cursor-up command
0	TTY	\<echo\	None
1	CRT	Erase	<VT>
2	CRT	Erase	<SUB>
3	CRT	Erase	~<FF>
4	CRT	Erase	<ESC>M

0 For TTY (hardcopy) terminals.

For such terminals, when the BACKSPACE, DEL, or RUBOUT key is pressed to begin a typing correction, a backslash character (\) is displayed along with the character that has been deleted. Subsequent character deletions cause just the deleted character to be displayed. When a nondeleting key is pressed, another backslash is displayed along with the new character. Thus, after the correction is completed, all the characters between the backslashes will have been deleted.

Another characteristic of this setting is that the WHERE 1 and IO commands will display their information only once.

If a CRT terminal is used with the appropriate one of the following settings, the WHERE 1 and IO commands will continuously update the display of their information until the user presses CTRL+C.

For all CRT terminals, when the BACK SPACE, DEL, or RUBOUT key is pressed, the last character input is deleted and erased from the screen.

1. For CRT terminals that accept an ASCII VT character (11 decimal) as the command to move the cursor up one line. (For example, use this value for Soroc brand terminals.)
2. For CRT terminals that accept an ASCII SUB character (26 decimal) as the command to move the cursor up one line. (For example, use this value for ADDS brand terminals.)
3. For CRT terminals that accept an ASCII ~ character (126 decimal) followed by an ASCII FF character (12 decimal) as the command to move the cursor up one line. (For example, use this value for Hazeltine brand terminals.)
4. For ANSI-compatible CRT terminals that accept an ASCII ESC character (27 decimal) followed by an ASCII M character (115 decimal) as the command to move the cursor up one line. (For example, use this value for VT100-compatible terminals, such as the Wyse terminal available from Adept.)

   This value should be used when the Adept graphics system is in use.

Example

Set system terminal type to VT100/Wyse:

```
PARAMETER TERMINAL = 4
```

Related Keywords

IO	(monitor command)
PROMPT	(program instruction)
TYPE	(program instruction)
WHERE	(monitor command)
PARAMETER	(monitor command, program instruction, and real-valued function)

See the ***V+ Operating System Reference Guide*** for details on monitor commands.

Syntax

```
TIME time_string
```

Function

Set the date and time.

Parameter

`time_string` String expression whose value specifies the date and time to be set. The value of the string must be in the format: dd-mmm-yy hh:mm:ss or dd-mmm-yy hh:mm (see below).

Details

The system clock is set equal to the value of the string expression.

The system clock is maintained automatically and should be changed only when its values are incorrect (e.g., the controller is moved to a different time zone).

The system clock is used in the following situations:

- The date and time are displayed when the V+ system is booted from disk.
- Whenever a new disk file is created, the date and time are recorded with the file name. (The FDIRECTORY command displays the dates and times for files.)
- The date and time are appended to the message indicating that an application program has terminated execution.
- The date and time are displayed by the TIME monitor command.
- The date and time are available to an application program by use of the $TIME string function.

The individual elements of the date and time specification are defined as follows:

dd	The day of the month (1 to 31)
mmm	The month, specified as a 3-letter abbreviation (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)
yy	The year, where 80 to 99 represent 1980 through 1999, respectively, and 00 to 79 represent 2000 through 2079, respectively.
hh	The hour of the day (0 to 23)
mm	Minutes past the hour (0 to 59)
ss	Seconds past the minute (0 to 59; 0 assumed if :ss omitted)

Example

```
TIME "23-JUN-83 16:10:25"
```

Related Keywords

TIME	(monitor command and real-valued function)
$TIME	(string function)

Syntax

TIME (string**, select)**

Function

Return an integer value representing either the date or the time specified in the given string parameter.

Parameters

`string` Optional string variable, constant, or expression that specifies the date and time in the format described below. (See below for details.)

`select` Real value, variable, or expression (interpreted as an integer) that selects the value to be returned. An error results if **`select`** is not one of the following:

select	Returned	Defined ss
1	date	(year-1980)*512 + month*32 + day
2	time	hour*2048 + minute*32 + second/2
3	seconds	time past the minute

Details

This function can be used to encode the date and time into compact (unsigned 16-bit) integer formats. After the integer date and time values are obtained, they can be arithmetically compared to other date and time values to determine before and after conditions.

> **NOTE:** You should not try to manipulate the encoded integer values to perform date or time arithmetic. For example, you should **not** attempt to add days to an encoded date value.

If the `string` parameter is supplied, both the date and the time must be specified in the string. The format must be dd-mmm-yy hh:mm:ss or dd-mmm-yy hh:mm. (The function returns the value –1 if the input string does not have an acceptable format [see the example below].)

The individual date and time elements are defined as follows:

dd	The day of the month (1 to 31)
mmm	The month, specified as a 3-letter abbreviation (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)
yy	The year, where 80 to 99 represent 1980 through 1999, respectively, and 00 to 79 represent 2000 through 2079, respectively.
hh	The hour of the day (0 to 23)
mm	Minutes past the hour (0 to 59)
ss	Seconds past the minute (0 to 59; 0 assumed if :ss omitted)

If the `string` parameter is not supplied and the **`select`** parameter is 1, the current date of the system clock is returned. In addition, the current time of the system clock is stored in the (internal) administrative data for the program task. If the `string` parameter is not supplied and the **`select`** parameter is 2 or 3, the selected time value is returned for the system-clock time *previously* saved.

Example

The following program segment shows how the TIME real-valued function can be used to make sure a valid date and time are entered by the user after a prompt:

```
PROMPT "Enter the date and time (dd-mmm-yy hh:mm:ss): ", $time

WHILE (TIME($time,1)== –1) DO            ;Make sure it's valid

TYPE /B,"    Could not interpret date/time."

PROMPT "Try again (dd-mmm-yy hh:mm:ss): ", $time

END

TIME $time                               ;Set system time
```

Related Keywords

TIME	(monitor command and program instruction)
$TIME	(string function)

Syntax

```
$TIME (date, time)
```

Function

Return a string value containing either the current system date and time or the specified date and time.

Parameters

`date` Optional integer value representing the year, month, and day (see below). The value is interpreted as follows (month ranges from 1 to 12):

```
date = (year-1980)*512 + month*32 + day
```

`time` Optional integer value representing the hour, minutes, and seconds past midnight (see below). The value is interpreted as follows (hour ranges from 0 to 23):

```
time = hour*2048 + minute*32 + second/2
```

NOTE: This function always returns a string containing both the date and the time. That can result in an erroneous date string if the `date` parameter is omitted when the `time` parameter is specified.

Details

If both the date and time parameters are omitted, this function returns the current system date and time in the format described below. (An empty string is returned if the system clock has not been initialized.)

If the date and time parameters are specified, their values are converted to an ASCII string in the format described below, and the string is returned. This operation is used to decode the output values generated by the TIME real-valued function.

The date and time are output in the format dd-mmm-yy hh:mm:ss, in which the individual elements are defined as follows:

dd	The day of the month (1 to 31)
mmm	The month, specified as a 3-letter abbreviation (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC)
yy	The year, where 80 to 99 represent 1980 through 1999, respectively, and 00 to 79 represent 2000 through 2079, respectively.
hh	The hour of the day (0 to 23)
mm	Minutes past the hour (0 to 59)
ss	Seconds past the minute (0 to 59)

NOTE: The $TIME function converts passed arguments (instead of the system time) when either the date or the time parameter is supplied. However, the function always tries to generate a string representation of **both** parameters. You get a strange date if you do not provide a date value. The time substring will be 00:00:00 if you do not specify a time value. The following expressions can be used to return only the date and the time, respectively:

```
$date = $MID($TIME(date,),1, 9)
$time = $MID($TIME(,time),11, 8)
```

Related Keyword

TIME	(monitor command, program instruction, and real-valued function)

Syntax

```
TIMER timer_number = time_value
```

Function

Set the specified system timer to the given time value.

Usage Considerations

Times measured by V+ are precise only to within 1 millisecond (0.001 seconds); shorter times cannot be measured.

System timer number 0 is the system clock and cannot be set with this instruction.

Parameters

`timer_number` Real-valued expression interpreted as the (integer) number of the timer to be set. The value must range from 1 to 15.

`timer_value` Real-valued expression interpreted as the time, in seconds, to which the timer is set. This parameter may specify fractions of a second and may be negative.

Details

When this instruction is executed, the specified timer is immediately set to the time specified. From then on, its value increases with time at the rate of 1 count per second. The timer will begin to lose accuracy after 4.7 hours have elapsed.

Use the TIMER real-valued function to read the instantaneous value of a system timer.

Example

The following examples show two ways to wait for a certain amount of time, using the TIMER instruction and real-valued function. Each example first sets the timer, and then waits until the timer value has changed by the delay period:

```
TIMER 1 = 0           ;Set timer to zero
WAIT TIMER(1) > delay ;Wait until timer > delay
TIMER 1 = -delay      ;Set timer to -delay
WAIT TIMER(1) > 0     ;Wait until timer > zero
```

Related Keyword

TIMER (real-valued function)

Syntax

```
TIMER (timer_number)
```

Function

Return the current time value (in seconds) of the specified system timer.

Usage Considerations

Times measured by V+ are precise only to within 1 millisecond (0.001 seconds); shorter times cannot be measured. The timer will begin to lose accuracy after 4.7 hours have elapsed.

Parameter

`timer_number` Real value, variable, or expression (interpreted as an integer) that specifies the number of the timer to be read. The value must be in the range –2 to 15.

Details

The TIMER function can be used to read a system timer at any time. The **`timer_number`** parameter selects which of the timers will be read.

Timers 1 through 15 can be set with the TIMER program instruction. Then, the value read later will represent the sum of the value last set with a TIMER instruction and the elapsed time (in seconds) since that instruction was executed. (See the example below.)

Timer # 0 is the system clock and records the number of seconds that have elapsed since the V+ system was started. (Timer number 0 cannot be set with the TIMER instruction.)

Before a setable timer is set with a TIMER instruction, it has the same value as timer number 0.

Timer # –1 provides a special interpretation of timer number 0. The function TIMER(–1) returns a 24-bit value that represents an integer value of 16-millisecond ticks (not seconds) of the system timer. Since the value from this TIMER function ranges from 0 to ^HFFFFFF, it can be used as a continuous, full-precision timer. Times greater than 74.5 hours will begin to lose accuracy.

Timer # –2 provides another special interpretation of timer number 0. The function TIMER(–2) returns a 24-bit value that represents an integer value of 1-millisecond ticks (not seconds) of the system timer. Since the value from this TIMER function ranges from 0 to ^HFFFFFF, it can be used as a continuous, full-precision timer. Times greater than 4.7 hours will begin to lose accuracy.

Timer #–3 provides the time in seconds since system startup. The function TIMER(–3) returns this time as a double-precision value. The time resolution is 1 millisecond. Because double-precision values have 52 bits of precision, the timer can run for more than 100,000 years before any precision is lost.

Example

The following example shows how the TIMER instruction and real-valued function can be used to time the execution of a subroutine:

```
TIMER 1 = 0          ;Set timer to zero
CALL test.routine() ;Call the subroutine
TYPE "Elapsed time =", TIMER(1)," seconds"
```

Related Keyword

TIMER (program instruction)

Syntax

```
TOOL transformation_value
```

Function

Set the internal transformation used to represent the location and orientation of the tool tip relative to the tool mounting flange of the robot.

Usage Considerations

The TOOL instruction causes a BREAK in continuous-path motion.

The TOOL instruction can be executed by any program task so long as the robot selected by the task is not attached by any other task. The instruction applies to the robot selected by the task.

If the V+ system is not configured to control a robot, executing the TOOL instruction will cause an error.

The word tool cannot be used as a program name or variable name.

Parameter

`transformation_value` Optional transformation variable or function, or compound transformation expression, that will be the new tool transformation. If the transformation value is omitted, the tool is set to NULL.

Details

Causes a BREAK in the robot continuous-path motion and sets the value of the tool transformation equal to the transformation value given.

Refer to the monitor TOOL command for a complete description of the effect of this instruction. (See the ***V+ Language User's Guide*** for information on how to define a tool transformation.)

Related Keywords

SELECT (program instruction and real-valued function)

TOOL (monitor command and transformation function)

Syntax

TOOL

Function

Return the value of the transformation specified in the last TOOL command or instruction.

Usage Considerations

The command LISTL TOOL can be used to display the current tool setting.

The TOOL function returns information for the robot selected by the task executing the function.

If the V+ system is not configured to control a robot, use of the TOOL function will not generate an error due to the absence of a robot. However, the information returned by the function may not be meaningful.

The name tool cannot be used as a program name or variable name.

Examples

Display the value of the current TOOL transformation from the system prompt:

```
LISTL TOOL
```

Save the value of the current TOOL:

```
SET save.tool = TOOL
```

Related Keywords

SELECT (program instruction and real-valued function)

TOOL (monitor command and transformation function)

Syntax

TPS

Function

Return the number of ticks of the system clock that occur per second (Ticks Per Second).

Usage Considerations

The name tps cannot be used as a program name or variable name.

Example

The following example shows how an event can be tested each system clock tick, with a time-out of 5 seconds, using the TPS function and the WAIT instruction.

```
FOR ticks = 1 TO 5*TPS ;Loop 5*ticks/sec times
   IF SIG(1001) THEN
     TYPE "Signal ON"
     HALT
   END

   WAIT                ;Wait until next clock tick

END

TYPE "Time-out while waiting for signal 1001"
```

Syntax

```
. . . TRACE
```

Function

Control the display of program steps on the system terminal during program execution.

Usage Considerations

Protected programs will not generate trace output.

TRACE applies only to program task #0. Thus, this switch does not affect programs executed as other program tasks.

Details

This system switch enables or disables a special mode of execution for program task #0, in which each program step is displayed on the system terminal before it is executed. This is useful while developing a program for checking the logical flow of execution.

Initially this switch is **disabled**.

Related Keywords

DISABLE	(monitor command and program instruction)
ENABLE	(monitor command and program instruction)
EXECUTE	(monitor command)
SSTEP	(monitor command)
SWITCH	(monitor command, program instruction, and real-valued function)
XSTEP	(monitor command)

See the *V+ Operating System Reference Guide* for details on monitor commands.

Syntax

```
TRANS (X_value, Y_value, Z_value, y_value, p_value, r_value)
```

Function

Return a transformation value computed from the given X, Y, Z position displacements and y, p, r orientation rotations.

Parameters

If any parameter is omitted, its value is taken to be zero.

`X_value`
`Y_value`
`Z_value`
	Optional expressions for the X, Y, and Z displacement components, respectively.

`y_value`
`p_value`
`r_value`
	Optional expressions for the yaw, pitch, and roll orientation components, respectively.

Details

The input parameter values are used to compute a transformation value that can be assigned to a location variable or used in a compound transformation or motion request.

Examples

If r is the radius of a circle and angle is the angle of rotation about the circle, then the transformation:

```
TRANS(r*COS(angle), r*SIN(angle), 0, 0, 0, 0)
```

will yield points on that circle.

If frame is a transformation defining the position of the center of the circle and the plane in which it lies, the following program segment will move the robot tool point around the circle in steps of 1 degree.

```
FOR angle = 0 TO 360-1
   MOVE frame:TRANS(r*COS(angle), r*SIN(angle), 0, 0, 0, 0)
END
```

Related Keywords

DECOMPOSE	(program instruction)
DX DY DZ	(real-valued functions)
#PPOINT	(precision-point function)
SET	(program instruction)
SHIFT	(program instruction)
TRANSB	(transformation function)

Syntax

TRANSB (string, first_char**)**

Function

Return a transformation value represented by a 48-byte string.

Parameters

string String expression that contains the 48 bytes to be converted.

first_char Optional real-valued expression that specifies the position of the first of the 48 bytes in the string.

If first_char is omitted or has a value of 0 or 1, the first 48 bytes of the string are extracted. If first_char is greater than 1, it is interpreted as the character position for the first byte. For example, a value of 2 means that the second through 49th bytes are extracted. An error is generated if first_char specifies 48 bytes that are beyond the end of the input string.

Details

Forty-eight sequential bytes of the given string are interpreted as being a set of twelve single-precision (32-bit) floating-point numbers in the IEEE standard format. (See the description of the FLTB function for details of the floating-point format.) The twelve values are interpreted as the components of a 3-by-4 transformation matrix, stored by column.

The main use of this function is to convert the binary representation of a transformation value from an input data record to values that can be used internally by V+.

Related Keywords

FLTB (real-valued function)

TRANS (transformation function)

$TRANSB (string function)

Syntax

```
$TRANSB (transformation)
```

Function

Return a 48-byte string containing the binary representation of a transformation value.

Parameter

`transformation` Transformation variable or function (or compound transformation) that defines the value to be converted to a string value.

Details

This function converts the given transformation value to the binary representation of its twelve (internal) components. The twelve values defining the transformation are the components of a 3-by-4 transformation matrix, stored by column. Each of the twelve 32-bit values is packed as four successive 8-bit characters in a string, resulting in a total of 48 characters. (The IEEE single-precision standard floating-point format is used for the conversion. See the FLTB real-valued function for a more detailed description of IEEE floating-point format.)

The main use of this function is to convert a transformation value to its binary representation in an output record of a data file.

Related Keywords

$FLTB (string function)

TRANSB (transformation function)

Syntax

```
... TRUE ...
```

Function

Return the value used by V+ to represent a logical true result.

Usage Considerations

The word true cannot be used as a program name or variable name.

Details

This named constant is useful for situations where true and false conditions need to be specified. The value returned is –1.

Example

The following program loop will execute continuously until the subroutine process returns a TRUE value for the real variable error:

```
DO
   CALL process(error)
UNTIL error == TRUE
```

The program loop below will execute indefinitely:

```
WHILE TRUE DO
   CALL move.part()
END
```

Related Keywords

FALSE (real-valued function)

ON (real-valued function)

Syntax

```
$TRUNCATE (string)
```

Function

Return all characters in the input string until an ASCII NUL (or the end of the string) is encountered.

Parameter

string	String variable, constant, or expression that specifies the string to be truncated.

Details

This function is similar to performing a $DECODE operation with an ASCII NUL (^H00) specified as the break character. $TRUNCATE differs from such a $DECODE operation in two ways:

- The input can be a string expression.
- The input string is not modified.

Because of its simplicity, the $TRUNCATE function executes much faster than the $DECODE function.

Example

The instruction below sets the value of the string variable $substring equal to abcdef. (Obviously, this is an artificial situation, since one would never want to perform a $TRUNCATE operation when the result is apparent from the input. However, it is presented to illustrate that this function can scan an arbitrary string expression and return the first substring delimited by a NUL.)

```
$substring = $TRUNCATE("abcdef"+$CHR(0)+"ghijk")
```

Related Keyword

$DECODE	(string function)

Syntax

```
TYPE output_specification, ..., output_specification
```

Function

Display the information described by the output specifications on the system terminal. A blank line is output if no argument is provided.

Usage Considerations

No output is generated if the MESSAGES system switch is disabled.

Program execution normally waits for the output to be completed before continuing. There is an output specification described below that can be used to prevent waiting if it is undesirable for execution to be delayed.

The output from a single TYPE instruction cannot exceed 512 characters. (The /S format control specifier described below can be used to output longer messages.)

Parameter

An `output_specification` can consist of any of the following components (in any order) separated by commas:

1. A string expression.
2. A real-valued expression, which is evaluated to determine a value to be displayed.
3. Format-control information, which determines the format of the output message.

Details

The following format-control specifiers can be used to control the way in which numeric values are displayed. These settings remain in effect for the remainder of the instruction, unless another specifier is used to change their effect.

For all these display modes, if a value is too large to be displayed in the specified field width, the field is filled with asterisk characters (*).

/D — Use the default format, which displays values to full precision with a single leading space. (Scientific notation is used for values greater than or equal to 1,000,000.)

NOTE: The following format specifications accept a zero as the field width (n). That causes the actual field size to vary to fit the value, and causes all leading spaces to be suppressed. That is useful when a value is displayed within a line of text or at the end of a line.

/En.m	Output values in scientific notation (for example, –1.234E+2) in fields n spaces wide with m digits the fractional parts. (If n is not 0 [see the note above], its value must be at least five larger than the value of m.)
/Fn.m	Output values in fixed-point notation (for example, –123.4) in fields n spaces wide, with m digits in the fractional parts.
/Gn.m	Output values in F format with m digits in the fractional parts if the values are larger than 0.01 and will fit in fields n spaces wide. Otherwise use /En.m format.
/Hn	Output values as hexadecimal integers in fields n spaces wide.
/In	Output values as decimal integers in fields n spaces wide.
/On	Output values as octal integers in fields n spaces wide.

The following specifiers can be used to control the appearance of the output.

/Cn	Output the characters carriage return (CR) and line feed (LF) n times. This will result in n blank lines if the control specifier is at the beginning or end of an output specification; otherwise, n–1 blank lines will result.
/S	Do not output a carriage return (CR) or line feed (LF) after displaying the current line.
/Un	Move the cursor up n lines. This will work correctly only if the TERMINAL parameter is correctly set for the terminal being used.
/Xn	Output n spaces.

The following specifiers can be used to perform control functions.

/B	Beep the terminal (nongraphics-based systems only).
/N	Initiate output without having program execution wait for its completion. A second output request will force program execution to wait for the first output if it has not yet completed.

Example

Assume that the real variable i has the value 5 and that array element point[5] has the value 12.666666. Then, the instruction

```
TYPE /B, "Point", i, " = " /F5.2, point[i]
```

will sound a beep at the system terminal (/B) and display the message

```
Point 5 = 12.67
```

If point[5] has the value 1000, the instruction will display

```
Point 5 = *****
```

because the value (1000.00) is too large to be displayed in the specified format (/F5.2). (The instruction would be able to display any value for point[5] if the format specification were /F0.5.)

Related Keywords

$ENCODE (string function)

MESSAGES (system switch)

PROMPT (program instruction)

WRITE (program instruction)

Syntax

UNIDIRECT directions[index**]**

Function

Specify that a joint is turning only in a single direction.

Usage Considerations

This instruction currently applies only to the JTS (joints) robot device module.

Parameter

directions A real-valued array that contains directional information for each configured joint in the robot. The element values are interpreted as follows:

Element value	Interpretation
1	The joint is configured for unidirectional travel in a positive joint-angle direction.
–1	The joint is configured for unidirectional travel in a negative joint-angle direction.
0	The joint has no special processing for unidirectional motion. This is the default state.

index Optional integer value identifying the array element that applies to joint #1. Zero is assumed if the index is omitted.

Details

For joints that are configured for continuous turning, this instruction can be used to specify when the joint is turning only in a single direction. Specifying unidirectional motion is important for synchronizing motion planning and the automatic rollover correction performed by the trajectory generator.

If a joint is specified as currently moving only in a positive direction, the motion planning routines automatically correct the specified destination joint angle such that the change in joint position (for any single motion) is less than the rollover amount. This eliminates the problem that can occur if a user program is generating a series of incremental motions and the trajectory generator asynchronously corrects the joint's position for rollover.

Once unidirectional mode is set, it remains in effect until another UNIDIRECT instruction is executed or the system is rebooted.

Example

Assume the following conditions:

1. Joint #2 is configured to roll over every 3600 degrees.
2. The joint has been configured for positive unidirectional motion with these instructions:

```
FOR jt = 1 TO ID(7, 8)          ;Set joints 1, 2, ...
  dir_array[jt] = 0             ;to default state
END

dir_array[2] = 1                ;Joint 2 isunidirectional

UNIDIRECT dir_array[1]          ;Apply the settings
```

3. The previous motion left the joint positioned at 3600 degrees.
4. The trajectory generator rolls the internal joint position back to 0 degrees. (The joint itself does not move.)
5. The next motion attempts to move the joint to 3700 degrees.

Given these conditions, the motion planner interprets the motion command in step 5 as though it were a command to the position 100 degrees.

By comparison, if the joint is not configured for unidirectional motion, the motion planner interprets the request to move the joint to 3700 degrees (step 5 above) as a request for a 3700-degree motion. That is, the joint moves 10 revolutions plus 100 degrees.

Syntax

$UNPACK (string_array[index**], first_char, num_chars)**

Function

Return a substring from an array of 128-character string variables.

Parameters

string_array String array variable from which the substring is to be extracted. It is assumed that each string within the array is defined and is 128 characters long.

index Optional integer value(s) that identifies the first array element to be considered. The **first_char** value is interpreted relative to the element specified by this index.

If no index is specified, element zero is assumed.

first_char Real-valued expression that specifies the position of the first character of the substring within the string array. A value of 1 corresponds to the first character of the specified string array element. This value must be greater than zero.

The value of **first_char** can be greater than 128. In that case the array element accessed will follow the element specified in the function call. For example, a value of 130 corresponds to the second character in the array element following that specified by index.

num_chars Real-valued expression that specifies the number of characters to be returned by the function. This value can range from 0 to 128.

Details

This function extracts a substring from an array of strings. Substrings are permitted to overlap two string array elements. For example, a 10-character substring whose first character is the 127th character in element [3] will be composed of the last two characters in element [3] followed by the first eight characters of element [4].

In order to efficiently access the string array, this function assumes that all of the array elements are defined and are 128 characters long. For multidimensional arrays, only the right-most array index is incremented to locate the substring. Thus, for example, element [2,3] is followed by element [2,4].

Example

The instruction below sets the value of the string variable $substring equal to a substring extracted from the string array $list[]. The substring is specified as starting in element $list[3]. However, since the first character is to be number 130, the 11-character substring will actually consist of the second through 12th characters of $list[4]:

```
$substring = $UNPACK($list[3], 130, 11)
```

Related Keywords

$MID (string function)

PACK (program instruction)

Syntax

```
UNTIL expression
```

Function

Indicate the end of a DO ... UNTIL control structure and specify the expression that is evaluated to determine when to exit the loop. The loop continues to be executed until the expression value is nonzero.

Usage Considerations

UNTIL must be used in conjunction with a DO control structure. See the description of the DO instruction for details.

Parameter

`expression` Real-valued expression, constant, or relation that is interpreted as either TRUE (nonzero) or FALSE (zero).

Details

If the expression in the UNTIL statement is zero, program execution continues with the statement following the matching DO statement. If the expression is nonzero, program execution continues with the statement following the UNTIL statement.

Example

The following example is a loop that continues to prompt the operator to enter a number until he/she enters one that is greater than or equal to zero:

```
DO
   PROMPT "Enter a positive number: ", number
UNTIL number >= 0
```

Related Keyword

DO (program instruction)

Syntax

```
... UPPER
```

Function

Control whether or not the case of each character is ignored when string comparisons are performed.

Details

When this switch is enabled and two strings are compared using the operators <, <=, ==, <>, >=, or >, all lowercase characters are treated as though they were uppercase characters. That is, when UPPER is enabled, both of the following comparisons will yield a TRUE value:

```
"a" == "A" and "A" == "A"
```

When UPPER is disabled, the case of characters is considered during string comparisons. Then, for example, the comparison on the left above would result in a FALSE value, while the comparison on the right would yield a TRUE value.

By default, UPPER is enabled, so that string comparisons are performed without considering the case of the characters.

Related Keywords

DISABLE (monitor command and program instruction)

ENABLE (monitor command and program instruction)

SWITCH (monitor command, program instruction, and real-valued function)

Syntax

VAL (string)

Function

Return the real value represented by the characters in the input string.

Usage Considerations

The input string can be a number in scientific notation.

The input string can contain leading number base indicators (^H, for example).

Any character that cannot be interpreted as part of a number or as a base indicator marks the end of the characters that are converted.

Parameter

string String constant, variable, or expression.

Examples

```
VAL("123 Elm Street") ;Returns the real value 123
VAL("1.2E-2")         ;Returns the real value 0.012
VAL("^HFF")           ;Returns the real value 255
```

Related Keywords

ASC (real-valued function)

$ENCODE (string function)

FLTB (real-valued function)

INTB (real-valued function)

LNGB (real-valued function)

Syntax

```
VALUE expression_list:
```

Function

Indicate the values that a CASE statement expression must match in order for the program statements immediately following to be executed.

Usage Considerations

VALUE must be part of a CASE control structure. See the description of the CASE instruction for details.

Parameter

`expression_list` List of real values or expressions separated by commas.

Related Keywords

ANY (program instruction)

CASE (program instruction)

Syntax

```
WAIT condition
```

Function

Put the program into await loop until the condition is TRUE.

Usage Considerations

An executing WAIT instruction in an application program can be canceled by using the PROCEED monitor command. WAIT can consume large amounts of CPU time.

Parameter

`condition`	Optional real value, variable, or expression that is tested for a TRUE (nonzero) or FALSE (zero) value.

Details

WAIT will suspend program execution until a desired condition exists. For example, the state of one or more external signals can be used for the condition for continuation.

If no `condition` is supplied, program execution is suspended until the next system cycle. System cycles occur at 16 millisecond intervals. A WAIT with no `condition` is especially useful in programs that need to perform an operation only once each system cycle.

If the WAIT condition is FALSE, V+ enters an internal loop, checking the condition and performing RELEASEs. It is equivalent to:

```
WHILE NOT condition DO
    RELEASE
END
```

> **NOTE:** If two high-priority tasks perform releases in the same time slice, they pass control back and forth to each other, effectively locking out any lower-priority tasks in the slice.

If you need to guarantee at least a 16-millisecond delay (for example, while manipulating signals monitored by REACT or REACTI), you should execute *two* consecutive WAIT instructions (with no arguments).

Examples

Stop program execution until external input signal #1001 is on and #1003 is off:

```
WAIT SIG(1001,-1003)
```

Stop program execution until the value of system timer #1 exceeds 10 (seconds). (This is a convenient way to introduce a delay in program execution.)

```
WAIT TIMER(1) > 10
```

Related Keywords

RELEASE (program instruction)

WAIT.EVENT (program instruction)

WAIT.START (monitor command, see the *V+ Operating System Reference Guide)*

Syntax

```
WAIT.EVENT mask, timeout
```

Function

Suspend program execution until a specified event has occurred, or until a specified amount of time has elapsed.

Usage Considerations

If a WAIT.EVENT instruction in an application program has execution suspended, the WAIT.EVENT can be canceled with the PROCEED monitor command.

Parameters

`mask` Optional real value, variable, or expression that specifies the events for which to wait. The value is interpreted as a sequence of bit flags, as detailed below. (All the bits are assumed to be clear if no mask value is specified.)

Bit 1 (LSB) Wait for I/O (mask value = 1)

If this bit is set, the desired event is the completion of any input/output operation by the current task.

`timeout` Optional real value, variable, or expression that specifies the number of seconds to wait. No time-out processing is performed if the parameter is omitted, or the value is negative or zero (see below for more details).

Details

This program instruction is used to suspend program execution until a specified event has occurred, or until a specified amount of time has elapsed. The program waits efficiently, because the occurrence of events and passage of time are checked by the V+ operating system without the program task executing. (See the example below for a comparison with the WAIT instruction.)

When the program resumes execution after a WAIT.EVENT instruction, the GET.EVENT function could be used to check if the desired event has actually occurred. This is the only way to distinguish between the occurrence of an event and a time-out (if one was specified).

If the `mask` parameter has the value zero (or is omitted), this instruction becomes a very efficient way to suspend program execution for the time period specified by the `timeout` parameter.

If the `timeout` parameter is omitted (or has a negative or zero value), this instruction suspends program execution indefinitely until the specified event occurs.

If both `mask` and`timeout` are zero or omitted, this instruction does nothing.

WAIT.EVENT 1 waits for an event to be signaled for a task. Events are signaled by either a SET.EVENT program instruction, or by a pending no-wait I/O instruction when the I/O operation is completed.

In general, there is no way to tell why the event was set. It may have been set by an I/O operation, a SET.EVENT program instruction, or an internal system process (such as a triggered REACT condition). For this reason, it is necessary to test for the desired condition after executing the WAIT.EVENT. For I/O, repeat the no-wait I/O operation or use the IOSTAT() function. For SET.EVENT issued by other tasks, define and check a global variable.

To avoid race conditions where the event is set or cleared between testing and waiting, use the following loop in the waiting task (the statement order is critical).

1. CLEAR.EVENT
2. Issue no-wait I/O if appropriate
3. Check I/O status or check global variable.
4. Exit loop if operation complete.
5. WAIT.EVENT 1
6. GOTO step 1

If using SET.EVENT to signal another task, use the following sequence (the statement order is critical).

1. Set the global variable
2. SET.EVENT for the appropriate task.

Examples

Suspend program execution (efficiently) for 5.5 seconds. In contrast, it is not efficient to use a WAIT instruction in conjunction with a system timer (as with the instructionWAIT TIMER(1) > 5.5), because the program task continuously executes, checking the value of the system timer:

```
WAIT.EVENT , 5.5
```

Suspend program execution until the completion of any system input/output, or until another program task sets events using the SET.EVENT instruction:

```
WAIT.EVENT 1
```

Suspend program execution for five seconds, until the completion of any system input/output, or until another program task uses the SET.EVENT instruction to set events. (The current program should use the GET.EVENT function to decide whether an event has occurred or five seconds has elapsed.)

```
WAIT.EVENT 1, 5
```

Related Keywords

CLEAR.EVENT (program instruction)

GET.EVENT (real-valued function)

INT.EVENT (program instruction)

SET.EVENT (program instruction)

Syntax

WHILE condition DO

Function

Initiate processing of a WHILE structure if the condition is TRUE or skipping of the WHILE structure if the condition is initially FALSE.

Usage Considerations

Every WHILE statement must be part of a complete WHILE ... DO ... END structure.

Parameter

condition Real-valued expression that is evaluated and tested for a TRUE (nonzero) or FALSE (zero) value.

Details

This structure provides another means for executing a group of instructions until a control condition is satisfied (compare it with the DO structure). The complete syntax for the WHILE structure is

```
WHILE condition DO
   group_of_steps
END
```

Processing of the WHILE structure can be described as follows:

1. Evaluate the **condition**. If the result is FALSE, proceed to item 4.
2. Execute the group_of_steps.
3. Return to item 1.
4. Continue program execution at the first instruction after the END step.

Unlike the DO structure described elsewhere, the group of instructions within the WHILE structure may not be executed at all. That is, if the condition has a FALSE value when the WHILE is first executed, then the group of instructions will not be executed at all.

When this structure is used, it is assumed that some action occurs within the group of enclosed instructions that will change the result of the logical expression from TRUE to FALSE when the structure should be exited.

Example

The following example uses a WHILE structure to monitor a combination of input signals to determine when a sequence of motions should be stopped. In this example, if the signal from either part feeder becomes zero (assumed to indicate the feeder is empty), then the repetitive motions of the robot will stop and the program will continue.

Note that if either feeder is empty when the WHILE structure is first encountered, then execution will immediately skip to step 27:

```
20 feeder.1 = 1037
21 feeder.2 = 1038
22 .
23 WHILE SIG(feeder.1, feeder.2) DO
24   CALL move.part.1()
25   CALL move.part.2()
26 END
27
28 ; Either feeder #1 or feeder #2 is empty
29 .
30 .
31 .
```

Related Keywords

DO (program instruction)

END (program instruction)

EXIT (program instruction)

NEXT (program instruction)

Syntax

```
WINDOW %belt_var = location, location, program, priority
```

Function

Set the boundaries of the operating region of the specified belt variable for conveyor tracking.

Usage Considerations

This option is available only if your Adept system is equipped with the V+ Extensions option.

The BELT switch must be enabled for this instruction to be executed.

The belt variable referenced must have already been defined using a DEFBELT instruction.

Parameters

`%belt_var` Name of the belt variable whose window is being established.

`location` Compound transformation that, together with the direction of the belt, defines one boundary of the operating window along the belt.

The window boundaries are planes that are perpendicular to the direction of belt travel and include the positions specified by the two transformations. The order of the transformations is not important—this instruction automatically determines which transformation represents the upstream boundary and which is for the downstream boundary.

`program` Optional program that is called if a window violation occurs while tracking the belt, subject to the specified priority level and the current priority level of the system.

`priority` Optional priority level of the window violation program. If no priority is specified, a priority of 1 is set.

Details

The operating window defined by this instruction is used both at motion planning time and motion execution time to determine if the destination of the motion is within acceptable limits.

When a motion is being planned, the destination of the motion is compared against the operating window. If a window violation occurs, the window violation program is ignored and a program error may be generated depending upon the setting of the BELT.MODE parameter and the nature of the error.

When a motion relative to the belt is being executed or after the motion is completed and the robot continues to track the destination, the destination is compared against the window every 16 milliseconds. If a window violation occurs and a program has been specified, the program is automatically invoked subject to its priority level, and the robot continues to track the belt and follow its continuous path motion. (The presumption is made that the specified program will direct the robot as required to recover from the window violation.)

If no program has been specified, the robot is immediately stopped and a window violation program error is signaled. If a REACTE has been posted, the REACTE routine will be activated. Otherwise, program execution will be terminated.

Example

The working window for the belt variable %belt1 is defined by locations win1 and win2. If a window violation ever occurs while the robot is tracking the belt, the program belt.error will be executed as a subroutine:

```
WINDOW %belt1 = win1, win2, belt.error
```

Related Keywords

BELT (system switch and real-valued function)

BELT.MODE (system parameter)

BSTATUS (real-valued function)

DEFBELT (program instruction)

SETBELT (program instruction)

WINDOW (real-valued function)

Syntax

WINDOW (transformation, time, mode**)**

Function

Return a value that indicates where the location described by the belt-relative transformation value is relative to the predefined boundaries of the working range on a moving conveyor belt.

Usage Considerations

This option is available only if your Adept system is equipped with the V+ Extensions option.

The BELT system switch must be enabled before this function can be used.

The belt variable referenced in the compound transformation must have already been defined using a DEFBELT instruction.

Parameters

transformation Compound transformation value that is defined relative to a belt. That is, the compound transformation must begin with a belt variable.

time Optional real-valued expression that specifies the time to look ahead when the transformation is evaluated. That is, the result of the function will be the value predicted to apply time seconds in the future, based on the current belt position and speed. This parameter is used to test if a motion can be correctly completed within an anticipated time period.

A time of zero (the default value) tests the instantaneous value of the location. Negative times are converted to 0 and times greater than 32,768/60 seconds are set equal to 32,768/60.

mode Optional real-valued expression that specifies whether the result of the function represents a distance inside or outside the belt window (see below).

Details

The value returned, which is a distance in millimeters, should be interpreted as described below. (Note that the definitions ofupstream anddownstream depend on the value of the BELT.MODE system parameter.)

1. If the value of the `mode` expression is **less than or equal to zero** (the default), the value returned is interpreted as follows (see **Figure 2-6**):

   0 The location is outside the window.

   <0 The location is inside the window, closest to the downstream window boundary; the distance is ABS(value_returned).

   >0 The location is inside the window, closest to the upstream window boundary; the distance to the boundary is the returned value.

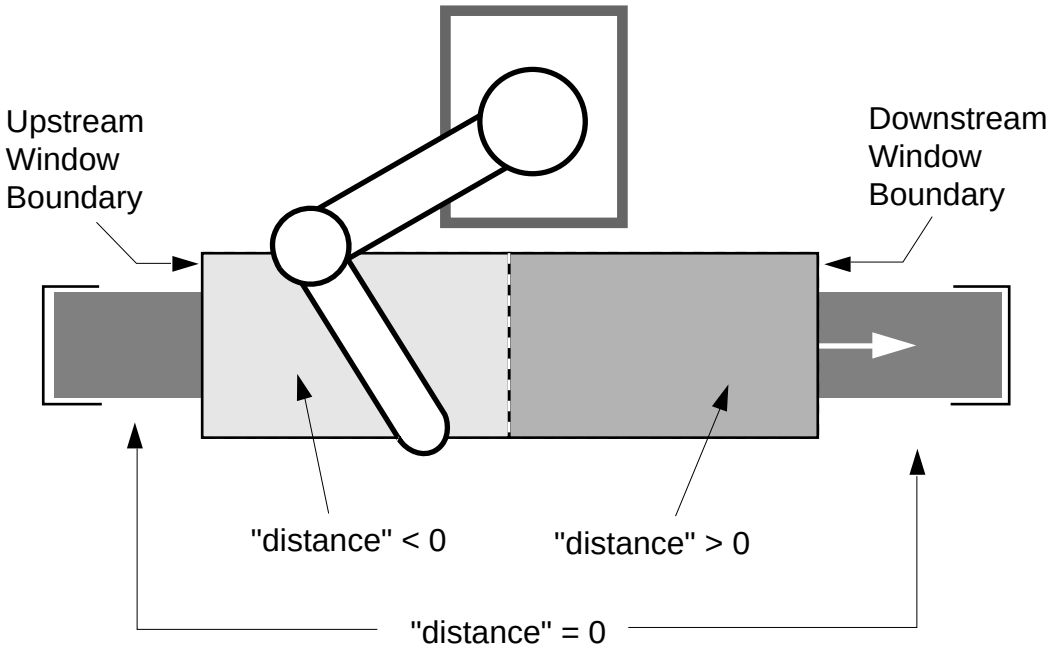


Figure 2-6. WINDOW Function for Mode Less Than or Equal to Zero

2. If the value of the `mode` expression is *greater than zero*, the value returned is interpreted as follows (see **Figure 2-7**):

   0 Indicates the location is within the window.

   <0 Indicates the location is upstream of the upstream window boundary; the distance is ABS(value_returned).

   >0 Indicates the location is downstream of the downstream window boundary; the distance is the value returned.

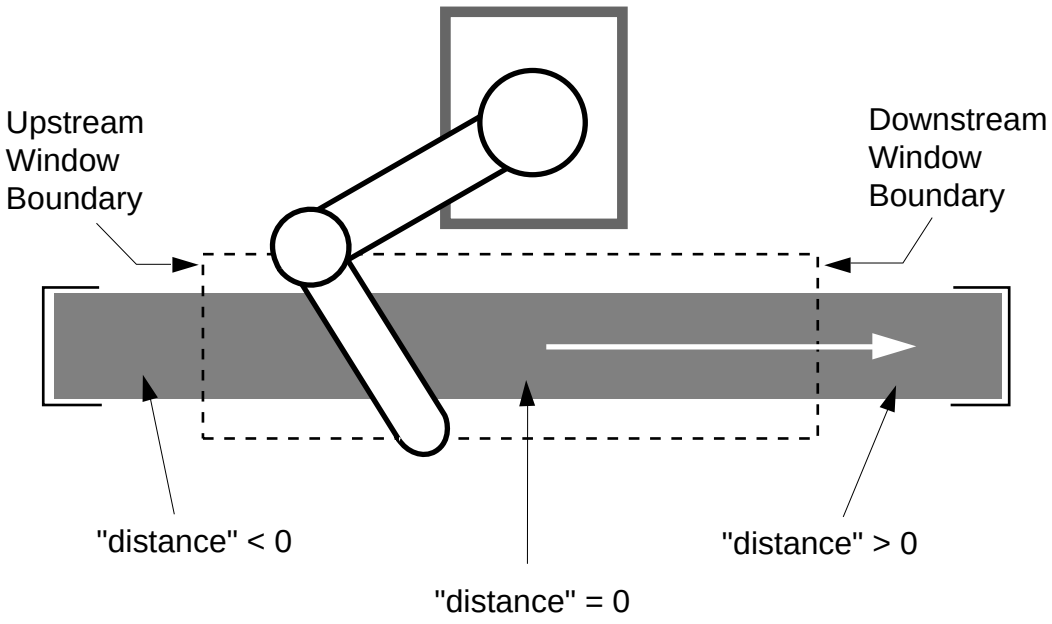


Figure 2-7. WINDOW Function for Mode Greater Than Zero

Note that the value returned by the WINDOW function always becomes more positive as the test location moves downstream (except for the discontinuity at the middle of the window when the mode value is less than or equal to zero).

Example

```
distance = WINDOW(%belt1:pick.up, 2, 1)
```

The distance will be nonzero if in two seconds the location will be outside the operating window for %belt1. Otherwise, distance will be zero if the location will be within the window.

Related Keywords

BELT (system switch and real-valued function)

BELT.MODE (system parameter)

BSTATUS (real-valued function)

DEFBELT (program instruction)

SETBELT (program instruction)

WINDOW (program instruction)

Syntax

```
WRITE (lun, record_num, mode) format_list
```

Function

Write a record to an open file, or to an attached device that is file oriented. For an AdeptNet device, write a string to an attached and open TCP connection.

Usage Considerations

The device to receive the output must have been attached. If the output is to a disk file, the file must have been opened with an FOPENA or FOPENW instruction.

Program execution waits for the write operation to complete unless there is a/N format specifier in the format list.

The AdeptNet aspects of this instruction apply only to Adept MV controllers with the AdeptNet Ethernet option and the AdeptTCP/IP Protocol Access option license.

Parameters

lun — Real-valued expression that identifies the device to be accessed. (See the ATTACH instruction for a description of unit numbers.)

record_num — Optional real-valued expression that represents the number of the record to be written. This should be 0 (the default value) to write the next sequential record. If the value is not zero, the record is written in random-access mode (which requires that the records all have the same length). In random-access mode, records are numbered from one (to a maximum of 16,777,216).

When accessing the TCP device with a server program, this parameter is an optional real value, variable, or expression (interpreted as an integer) that defines the client handle. For more information, refer to documentation for the READ instruction and to the ***AdeptNet User's Guide***.

mode — Optional real-valued expression that represents the mode of operation. At present, this value is not used and can be omitted.

format_list Consists of a list of output variables, string expressions, and format specifiers used to create the output record. The format list is processed exactly like an output specification for the TYPE instruction.

When accessing the TCP device, you can include the /N specifier to prevent the V+ system from waiting for a write acknowledgment.

Details

This is a general-purpose data output instruction that writes a record to a specified logical unit. A record can contain an arbitrary list of characters, but must not exceed 512 characters in length.

For files that are opened in fixed-length record mode, this instruction appends NUL characters to the output record if it is shorter than the file records.

When accessing the TCP/IP device, the record_num parameter enables a server to communicate with multiple clients on a single logical unit. Handles are allocated when a client connects to the server and then deallocated when a client disconnects. During a connection the READ instruction that receives data from the TCP logical unit returns the client handle. A WRITE instruction then can use the handle value to send data to the corresponding client. Refer to the example program in the *AdeptNet User's Guide*.

Examples

You can write a message to the manual control pendant with the following instructions:

```
ATTACH (1)          ;Attach the control pendant
WRITE (1) $message  ;Output message to pendant
DETACH (1)          ;Detach the pendant
```

A file with variable-length records can be written to the system disk drive with instructions such as the following:

```
ATTACH (dlun, 4) "DISK"       ;Attach the disk interface
FOPENW (dlun) "A:testfile.dat";Open a file on drive "A"
FOR i = 0 TO LAST($lines[])   ;Loop for all the elements
   WRITE (dlun) $lines[i]     ;to be written
END
FCLOSE (dlun)                 ;Close the file
DETACH (dlun)                 ;Release the disk interface
```

Attach to serial line 1 and write a greeting:

```
ATTACH (slun, 4) "SERIAL:1"
WRITE "Hello from serial line 1"
```

Write the string `$str` to the client defined by the handle, which must have been defined previously when the a message was received. Do not wait for acknowledgment:

```
WRITE (lun, handle) $str, /N
```

Related Keywords

ATTACH (program instruction)

DETACH (program instruction)

FCLOSE (program instruction)

FEMPTY (program instruction)

FOPEN_ (program instructions)

IOSTAT (real-valued function)

Syntax

```
... value XOR value ...
```

Function

Perform the **logical** exclusive-OR operation on two values.

Details

The XOR operator operates on two values, resulting in their logical exclusive-OR combination. For example, during the exclusive-OR operation

```
c = a XOR b
```

the following four situations can occur:

a	b		c
FALSE	FALSE	➡	FALSE
FALSE	TRUE	➡	TRUE
TRUE	FALSE	➡	TRUE
TRUE	TRUE	➡	FALSE

That is, the result is TRUE if only **one** of the two operand values is logically TRUE.

Refer to the *V+ Language User's Guide* for the order in which operators are evaluated within expressions.

Example

In the following sequence, the instructions immediately following the IF instruction will be executed if ready is TRUE (that is, nonzero) or count equals 1. The instructions will not be executed if both ready is FALSE and count is not equal to 1, or if ready is TRUE and count equals 1.

```
IF ready XOR (count == 1) THEN
.
.
.
END
```

Related Keywords

AND (operator)

BXOR (operator)

OR (operator)

V+ Language Quick Reference A

ABORT `task_num`
Terminate execution of an executing program task.

ABOVE
Request a change in the robot configuration during the next motion so that the elbow is above the line from the shoulder to the wrist.

ABS **`(value)`**
Return absolute value.

ACCEL `(profile) acceleration, deceleration`
Set acceleration and deceleration for robot motions. Optionally, specify a defined acceleration profile.

ACCEL **`(select)`**
Return the current setting for robot acceleration or deceleration setting or return the maximum allowable percentage limits set by the SPEC utility program.

AIO.IN **`(channel`**`, gain`**`)`**
Read a channel from one of the analog IO boards.

AIO.INS **`(channel)`**
Test whether an analog input or output channel is installed.

AIO.OUT **`channel = value`**
Write to a channel on one of the analog IO boards.

ALIGN
Align the robot tool Z axis with the nearest world axis.

ALTER **`(`**`control`**`)`** `Dx, Dy, Dz, Rx, Ry, Rz`
Specify the magnitude of the real-time path modification that is to be applied to the robot path during the next trajectory computation.

ALTOFF
Terminate real-time path-modification mode ("alter" mode).

ALTON **`(`**`lun`**`)`** `mode`
Enable real-time path-modification mode ("alter" mode), and specify the way in which ALTER coordinate information will be interpreted.

ALWAYS
Used with certain program instructions to specify a long-term effect.

AND **`value ...`**
Perform the *logical* AND operation on two values.

ANY
Signal the beginning of an alternative group of instructions for the CASE structure.

APPRO **`location, distance`**
APPROS **`location, distance`**
Start a robot motion toward a location defined relative to specified location.

ASC **`(string`**`, index`**`)`**
Return an ASCII character value from within a string.

ATAN **`(value_, value_)`**
Return the size of the angle (in degrees) that has its trigonometric tangent equal to value_/value_.

ATTACH **`(`**`lun, mode`**`)`** `$device`
Make a device available for use by the application program.

AUTO `type` **`variable`**`, ..., variable`
Declare temporary variables that are automatically created on the program stack when the program is entered.

AUTO.POWER.OFF
Control whether or not V+ disables high power when certain motion errors occur.

BAND **`value ...`**
Perform the binary AND operation on two values.

BASE X_shift, Y_shift, Z_shift, Z_rotation
Translate and rotate the World reference frame relative to the robot.

BASE
Return the transformation value that represents the translation and rotation set by the last BASE command or instruction.

BCD **(value)**
Convert a real value to Binary Coded Decimal (BCD) format.

BELOW
Request a change in the robot configuration during the next motion so that the elbow is below the line from the shoulder to the wrist.

BELT
Control the function of the conveyor tracking features of the V+ system.

BELT **(%belt_var**, mode**)**
Return information about a conveyor belt being tracked with the conveyor tracking feature.

BELT.MODE
Set characteristics of the conveyor tracking feature of the V+ system.

BITS **first_sig**, num_sigs **= value**
Set or clear a group of digital signals based on a value.

BITS **(first_sig**, num_sigs**)**
Read multiple digital signals and return the value corresponding to the binary bit pattern present on the signals.

BMASK **(bit**, bit, ..., bit**)**
Create a bit mask by setting individual bits.

BOR **value ...**
Perform the binary OR operation on two values.

BRAKE
Abort the current robot motion.

BREAK
Suspend program execution until the current motion completes.

BSTATUS
Return information about the status of the conveyor tracking system.

BXOR **value ...**
Perform the binary exclusive-OR operation on two values.

BY (value)
BY (value, value, value)
Complete the syntax of the SCALE and SHIFT functions.

CALIBRATE mode, status
Initialize the robot positioning system with the robot's current position.

CALL **program(**arg_list**)**
Suspend execution of the current program and continue execution with a new program (that is, a subroutine).

CALLP **var(arg_list)**
Call a program given a pointer to the program in memory.

CALLS **string(**arg_list**)**
Suspend execution of the current program and continue execution with a new program (that is, a subroutine) specified with a string value.

CASE **value** OF
Initiate processing of a CASE structure by defining the value of interest.

$CHR **(value)**
Return a one-character string corresponding to a given ASCII value.

CLEAR.EVENT **task**, flag, processor
Clear an event associated with the specified task.

CLOSE
CLOSEI
Close the robot gripper.

COARSE tolerance ALWAYS
Enable a low-precision feature of the robot hardware servo.

COM **value ...**
Perform the binary complement operation on a value.

CONFIG **(**select**)**
Return a value that provides information about the robot's geometric configuration, or the status of the motion servo-control features.

COS **(value)**
Return the trigonometric cosine of a given angle.

CP
Control the continuous-path feature.

CPOFF ALWAYS
Instruct the V+ system to stop the robot at the completion of the next motion instruction (or all subsequent motion instructions) and null

position errors.

CPON ALWAYS
Instruct the V+ system to execute the next motion instruction (or all subsequent motion instructions) as part of a continuous path.

CYCLE.END task_num, stop_flag
Terminate the executing program in the specified task the next time it executes a STOP program instruction (or its equivalent).

Suspend processing of an executable program until a program running in the specified task completes execution.

DBLB **($string**, first_char**)**
Return the value of eight bytes of a string interpreted as an IEEE double-precision floating-point number.

$DBLB **(value)**
Return an 8-byte string containing the binary representation of a real value in double-precision IEEE floating-point format.

DCB **(value)**
Convert BCD digits into an equivalent integer value.

DECEL.100 [robot_num]
Enable or disable the use of 100 percent as the maximum deceleration for the ACCEL program instruction.

$DECODE **($string_var, string_exp,** mode**)**
Extract part of a string as delimited by given break characters.

DECOMPOSE **array_name[**index**] = location**
Extract the (real) values of individual components of a location value.

$DEFAULT **()**
Return a string containing the current system default device, unit, and directory path for disk file access.

DEFBELT **%belt_var = nom_trans, belt_num, vel_avg, scale_fact**
Define a belt variable for use with a conveyor tracking robot.

DEF.DIO **signal = address,** type
Assign third-party digital I/O boards to standard V+ signal numbers, for use by standard V+ instructions, functions, and monitor commands.

DEFINED **(var_name)**
Determine whether a variable has been defined.

DELAY **time**
Cause robot motion to stop for the specified period of time.

DEPART **distance**
DEPARTS **distance**
Start a robot motion away from the current location.

DEST
Return a transformation value representing the planned destination location for the current robot motion.

DETACH (logical_unit)
Release a specified device from the control of the application program.

DEVICE **(type, unit,** error, p, p, ...**)** out[i], in[j], out_trans, in_trans
Send a command or data to an external device and, optionally, return data back to the program. (The actual operation performed depends on the device referenced.)

DEVICE **(type, unit,** error, p, p,...**)**
Return a real value from a specified device. The value may be data or status information, depending upon the device and the parameters.

DEVICES **(type, unit,** error, p, p, ...**)** $out, $in
Send commands or data to an external device and optionally return data. The actual operation performed depends on the device referenced.

DISABLE **switch,** ..., switch
Turn off one or more system control switches.

DISTANCE **(location, location)**
Determine the distance between the points defined by two location values.

DO
Introduce a DO program structure.

DOS **string,** error
Execute a program instruction defined by a string expression.

DRIVE **joint, change, speed**
Move an individual joint of the robot.

DRY.RUN
Control whether or not V+ communicates with the robot.

DURATION **time** ALWAYS
Set the minimum execution time for subsequent robot motions.

DURATION **(select)**
Return the current setting of one of the motion DURATION specifications.

DX **(location)**
DY **(location)**
DZ **(location)**
Return a displacement component of a given transformation value.

ELSE
Separate the alternate group of statements in an IF ... THEN control structure.

ENABLE **switch**, ..., switch
Turn on one or more system control switches.

$ENCODE **(output_specification,** output_specification, ...**)**
Return a string created from output specifications. The string produced is similar to the output of a TYPE instruction.

END
Mark the end of a control structure.

.END
Mark the end of a V+ program.

ERROR **(task_num**, select**)**
Return the error number of a recent error that caused program execution to stop or caused a REACTE reaction.

$ERROR **(error_code)**
Return the error message associated with the given error code.

ESTOP
Assert the emergency-stop signal to stop the robot.

EXECUTE /C task_num **program(**param_list**)**, cycles, step, priority[i]
Begin execution of a control program.

EXIT count
Branch to the statement following the nth nested loop of a control structure.

FALSE
Return the value used by V+ to represent a logical false result.

FCLOSE **(logical_unit)**
Close the disk file, graphics window, or graphics icon currently open on the specified logical unit.

FCMND **(logical_unit, command_code)** **$out_string**, $in_string
Generate a device-specific command to the input/output device specified by the logical unit.

FDELETE **(logical_unit) object**
Delete the specified disk file, the specified graphics window and all its child windows, or the specified graphics icon.

FEMPTY **(logical_unit)**
Empty any internal buffers in use for a disk file or a graphics window by writing the buffers to the file or window if necessary.

FINE tolerance ALWAYS
Enable a high-precision feature of the robot hardware servo.

FLIP
Request a change in the robot configuration during the next motion so that the pitch angle of the robot wrist has a negative value.

FLTB **($string**, first_char**)**
Return the value of four bytes of a string interpreted as an IEEE single-precision floating-point number.

$FLTB **(value)**
Return a 4-byte string containing the binary representation of a real value in single-precision IEEE floating-point format.

FOPEN **(logical_unit,** mode**)** **attribute_list**
Create and open a new graphics window or TCP connection, or open an existing graphics window for subsequent input or output.

FOPEN_ **(lun**, record_len, mode**)** **file_spec**
Open a disk file for read-only, read-write, read-write-append, or read-directory, respectively, as indicated by the last letter of the instruction name.

FOR **loop_var = initial TO final** STEP increment
Execute a group of program instructions a certain number of times.

FORCE._
AdeptForce option status and control instructions.

FRACT **(value)**
Return the fractional part of the argument.

FRAME **(location_, location_, location_, location_)**
Return a transformation value defined by four positions.

FREE (memory, select)
Return the amount of unused free memory storage space.

FSEEK **(logical_unit,** record_number)
Position a file open for random access and initiate a read operation on the specified record.

FSET **(logical_unit) attribute_list**
Set or modify attributes of a graphics window, serial line, or network device related to AdeptNet.

GAIN.SET set, motor
Select a set of feedback gain parameters for one or more motors of the currently selected robot.

GARC **(lun,** mode**) xc, yc, radius,** ang, angn
Draw an arc or a circle in a graphics window.

GCHAIN **(lun)** x, y, **points, direction[**index**]**
Draw a chain of points in a graphics window to form a complex figure.

GCLEAR **(lun)**
Clear an entire graphics window to the background color.

GCLIP **(lun)** x, y, dx, dy
Set the clipping rectangle for all graphics instructions (except GFLOOD), to suppress all subsequent graphics that fall outside the rectangle.

GCOLOR **(lun)** foregrnd, backgrnd
Set the foreground and background colors for subsequent graphics output.

GCOPY **(lun) x, y = src_x, src_y, dx, dy**
Copy one region of a window to another region in the same window.

GET.EVENT **(task)**
Return events that are set for the specified task.

GETC **(lun, mode)**
Return the next character (byte) from a device or input record on the specified logical unit.

GETEVENT **(lun,** mode**) events[**index**]**
Return information describing input from a graphics window or input from the terminal.

GFLOOD **(logical_unit) x, y**
Flood a region in a graphics window with color.

GGETLINE **(logical_unit) $data[**index**],** num.pix **= x, y, nx**
Return pixel information from a single pixel row in a graphics window.

GGET.LINE **(logical_unit) $data[**index**],** num.pix **= x, y, nx**
Return pixel information from a single pixel row in a graphics window.

GICON **(lun,** mode**) x, y, $name,** index
Draw a predefined graphic symbol (icon) in a graphics window.

GLINE **(lun) x, y, xn, yn**
Draw a single line segment in a graphics window.

GLINES **(logical_unit,** mode**) points, coord[**offset,index**]**
Draw multiple line segments in a graphics window.

GLOBAL type **variable,** ..., variable
Declare a variable to be global and specify the type of the variable.

GLOGICAL **(logical_unit)** code, planes
Set the logical operation to be performed between new graphics output and graphics data already displayed, and select which bit planes are affected by graphics instructions.

GOTO **label**
Perform an unconditional branch to the program step identified by the given label.

GPANEL **(lun,** mode**) x, y, dx, dy**
Draw a rectangular panel with shadowed or grooved edges.

GPOINT **(lun) x, y**
Draw a single point in a graphics window.

GRECTANGLE **(lun,** mode**) x, y, dx, dy**
Draw a rectangle in a graphics window.

GSCAN **(lun) lines, data[**offset,index**]**
Draw a number of horizontal lines in a graphics window to form a complex figure.

GSLIDE **(lun**, mode**) id =** x, y, **length, max_pos, arrow_inc**, handle
Draw a slide bar in preparation for receiving slide events.

GTEXTURE **(lun)** mode, pattern
Set the opaque/transparent mode and the texture pattern for subsequent graphics output.

GTRANS **(lun**, mode**)** array[,]
Scale, rotate, offset, and apply perspective correction to all subsequent graphics instructions.

GTYPE **(lun**, mode**) x, y, $text**, font_numb, path, rotation
Display a text string in a graphics window.

HALT
Stop program execution and do not allow the program to be resumed.

HAND
Return the current hand opening.

HAND.TIME
Establish the duration of the motion delay that occurs during OPENI, CLOSEI, and RELAXI instructions.

HERE **location_var**
Set the value of a transformation or precision point variable equal to the current robot location.

HERE
Return a transformation value that represents the current location of the robot tool point.

HOUR.METER
Return the current value of the robot hour meter.

ID **(component**, device, board**)**
Return values that identify the configuration of the current system.

$ID **(select)**
Return the system ID string.

IDENTICAL **(location, location)**
Determine if two location values are exactly the same.

IF GOTO **logical_expr label**
Branch to the specified label if the value of a logical expression is TRUE (nonzero).

IF **logical_expr** THEN
Conditionally execute a group of instructions (or one of two groups) depending on the result of a logical expression.

IGNORE **signal**
Cancel the effect of a REACT or REACTI instruction.

INRANGE **(location)**
Return a value that indicates if a location can be reached by the robot, and if not, why not.

INSTALL **password,** op
Install or remove software options available to Adept systems.

INT **(value)**
Return the integer part of the value.

INTB **($string**, first_char**)**
Return the value of two bytes of a string interpreted as a signed 16-bit binary integer.

$INTB **(value)**
Return a 2-byte string containing the binary representation of a 2-bit integer.

INTERACTIVE
Control the display of message headers on the system terminal and requests for confirmation before performing certain operations.

INT.EVENT **source**, level
Send an event (as though from a SET.EVENT instruction) to the current task if an interrupt occurs on a specified VMEbus vector or a specified digital I/O signal transitions to positive.

INVERSE **(transformation)**
Return the transformation value that is the mathematical inverse of the given transformation value.

IOGET_ **(address**, type, cpu**)**
Return a value from global memory or from a device on the VME bus.

$IOGETS **(address, length**, type, cpu**)**
Return a string value from a device on the VME bus.

IOPUT_ **address**, type, cpu **= value**
Write a value to global CPU memory or to a device on the VME bus.

IOSTAT **(lun**, mode**)**
Return status information for the last input/output operation for a device associated with a logical unit.

IOTAS **(address**, type, cpu**)**
Control access to shared devices on the VME bus.

IPS ALWAYS
Specify the units for a SPEED instruction as inches per second.

KERMIT.RETRY
Establish the maximum number of times the (local) Kermit driver should retry an operation before reporting an error.

KERMIT.TIMEOUT
Establish the delay parameter that the V+ driver for the Kermit protocol will send to the remote server.

KEYMODE **first_key**, last_key **= mode**, setting
Set the behavior of a group of keys on the manual control pendant.

KILL task_number
Clear a program execution stack and detach any I/O devices that are attached.

LAST **(array_name[])**
Return the highest index used for an array (dimension).

LATCH **(select)**
Return a transformation value representing the location of the robot at the occurrence of the last external trigger or AdeptForce guarded-mode trigger.

LATCHED **(select)**
Return the status of the external trigger and/or an AdeptForce guarded-mode trigger.

LEFTY
Request a change in the robot configuration during the next motion so that the first two links of a SCARA robot resemble a human's left arm.

LEN **(string)**
Return the number of characters in the given string.

LNGB **($string**, first_char**)**
Return the value of four bytes of a string interpreted as a signed 32-bit binary integer.

$LNGB **(value)**
Return a 4-byte string containing the binary representation of a 4-bit integer.

LOCAL type **variable**, ..., variable
Declare permanent variables that are defined only within the current program.

LOCK **priority**
Set the program reaction lock-out priority to the value given.

MAX **(value**, ..., value**)**
Return the maximum value contained in the list of values.

MC **monitor_command**
Introduce a monitor command within a command program.

MCP.MESSAGE
Conrol how system error messages are handled when the controller keyswitch is *not* in the MANUAL position.

MCS **string**
Invoke a monitor command from an application program.

MCS.MESSAGE
Enable or disable output to the system terminal from monitor commands executed with the MCS instruction.

MESSAGES
Enable or disable output to the system terminal from TYPE instructions.

$MID **(string**, first_char, **num_chars)**
Return a substring of the specified string.

MIN **(value**, ..., value**)**
Return the minimum value contained in the list of values.

MMPS ALWAYS
Specify the units for a SPEED instruction as millimeters per second.

MOD **value** ...
Compute the modulus of two values.

MONITORS
Enable or disable selecting of multiple monitor windows.

MOVE **location**
MOVES **location**
Initiate a robot motion to the position and orientation described by the given location.

MOVEF **location**, depart_clr, appro_clr, **depart_tqe**, **horiz_accel_tqe**, **horiz_decel_tqe**, **appro_tqe**, **model**

MOVESF **location**, depart_clr, appro_clr, **depart_tqe**, **horiz_accel_tqe**,

horiz_decel_tqe, appro_tqe, model
Initiate a three-segment pick-and-place robot motion to the specified destination, moving the robot at the fastest allowable speed.

MOVET **location, hand_opening**
MOVEST **location, hand_opening**
Initiate a robot motion to the position and orientation described by the given location and simultaneously operate the hand.

MULTIPLE ALWAYS
Allow full rotations of the robot wrist joints.

NETWORK **(component,**code**)**
Return network status and IP address information.

NEXT count
Branch to the END statement of the nth nested loop, perform the loop test, and loop if appropriate.

NOFLIP
Request a change in the robot configuration during the next motion so that the pitch angle of the robot wrist has a positive value.

NONULL ALWAYS
Instruct the V+ system not to wait for position errors to be nulled at the end of continuous-path motions.

NOOVERLAP ALWAYS
Generate a program error if a motion is planned that will cause selected multiturn axes to turn more than ±180 degrees (the "long way around") in order to avoid a limit stop.

NORMAL **transformation_value)**
Correct a transformation for any mathematical round-off errors.

NOT value ...
Perform logical negation of a value.

NOT.CALIBRATED
Indicate (or assert) the calibration status of the robots connected to the system.

NULL ALWAYS
Instruct the V+ system to wait for position errors to be nulled at the end of continuous path motions.

NULL
Return a null transformation value—one with all zero components.

OFF
Return the value used by V+ to represent a logical false result.

ON
Return the value used by V+ to represent a logical true result.

OPEN
OPENI
Open the robot gripper.

OR **value ...**
Perform the logical OR operation on two values.

OUTSIDE **(low, test, high)**
Test a value to see if it is outside a specified range.

OVERLAP ALWAYS
Disable the NOOVERLAP limit-error checking either for the next motion or for all subsequent motions.

PACK **string_array[**index**],**
first_char, num_chars = string
PACK **string_var, first_char,**
num_chars = string
Replace a substring within an array of (128-character) string variables, or within a (nonarray) string variable.

PARAMETER **parameter_name = value**
PARAMETER **parameter_name[**index**] =**
value
Set the value of a system parameter.

PARAMETER **(parameter_name)**
PARAMETER **(parameter_name[index])**
Return the current setting of the named system parameter.

PAUSE
Stop program execution, but allow the program to be resumed.

PAYLOAD value, motor
Adjust the feedforward compensation for a specified motor by setting a percentage of the maximum payload assumed for that motor.

#PDEST

Return a precision-point value representing the planned destination location for the current robot motion.

PENDANT **(select)**
Return input from the manual control pendant.

PI
Return the value of the mathematical constant pi.

#PLATCH (select)
Return a precision-point value representing the location of the robot at the occurrence of the last external trigger or AdeptForce guarded-mode trigger.

POS (search_string, sub_string, start)
Return the starting character position of a substring in a string.

POWER
Control or monitor the status of high power.

#PPOINT (j_value, j_value, j_value, j_value, j_value, j_value)
Return a precision-point value composed from the given components.

PRIORITY
Return the current reaction lock-out priority for the program.

.PROGRAM
program_name(argument_list) ;comment
Define the arguments a program will be passed when it is invoked.

PROMPT output_string, **variable_list**
Display a string on the system terminal and wait for operator input.

RANDOM
Return a pseudorandom number.

REACT **signal_num, program**, priority
Initiate continuous monitoring of a specified digital signal and automatically trigger a subroutine call if the signal properly transitions.

REACTE **program_name**
Initiate the monitoring of errors that occur during execution of the current program task.

REACTI **signal_num,** program, priority
Initiate continuous monitoring of a specified digital signal. Automatically stop the current robot motion if the signal transitions properly and optionally trigger a subroutine call.

READ (lun, record_num, mode) **var_list**
Read a record from an open file or from an attached device that is not file oriented. For an AdeptNet device, read a string from an attached and open TCP connection.

READY
Move the robot to the READY location above the workspace, which forces the robot into a standard configuration.

RELAX
RELAXI
Limp the pneumatic hand.

RELEASE task
Allow the next available program task to run.

RESET
Turn "off" all the external output signals.

RETRY
Control whether the PROGRAM START button causes a program to resume.

RETURN
Terminate execution of the current subroutine, and resume execution of the last-suspended program at the step following the CALL or CALLS instruction that caused the subroutine to be invoked.

RETURNE
Terminate execution of an error reaction subroutine and resume execution of the last-suspended program at the step following the instruction that caused the subroutine to be invoked.

RIGHTY
Request a change in the robot configuration during the next motion so that the first two links of the robot resemble a human's right arm.

ROBOT [index]
Enable or disable one robot or all robots.

ROBOT.OPR Execute operations that are specific to the currently selected robot or robot module.

RUNSIG signal_num
Turn on (or off) the specified digital signal as long as execution of the invoking program task continues.

RX (angle)
RY (angle)
RZ (angle)
Return a transformation describing a rotation.

SCALE (transformation BY **scale_factor)**
Return a transformation value equal to the transformation parameter with the position scaled by the scale factor.

SCALE.ACCEL [robot_num]
Enable or disable the scaling of acceleration and deceleration as a function of program speed when program speed is below a preset value.

SCALE.ACCEL.ROT [robot_num]
Specify whether or not the SCALE.ACCEL switch takes into account the Cartesian rotational speed during straight-line motions.

SCREEN.TIMEOUT
Establish the time-out period for blanking the screen of the graphics monitor.

SEE **(lun) prog_spec**, step
Invoke the screen-oriented program editor to allow a program to be created, viewed, or modified.

SELECT **device_type = unit**
Select a unit of the named device for access by the current task.

SELECT **(device_type**, mode**)**
Return the unit number that is currently selected by the current task for the device named.

SET **location_var = location_value**
Set the value of the location variable on the left equal to the location value on the right of the equal sign.

SET.EVENT **task**, flag, processor
Set an event associated with the specified task.

#SET.POINT
Return the commanded joint-angle positions computed by the trajectory generator during the last trajectory-evaluation cycle.

SET.SPEED
Control whether or not the monitor speed can be changed from the manual control pendant. The monitor speed cannot be changed when the switch is disabled.

SETBELT **%belt_var = expression**
Set the encoder offset of the specified belt variable equal to the value of the expression.

SETDEVICE **(type, unit,** error, **command)** p, p, ...
Initialize a device or set device parameters. (The actual operation performed depends on the device referenced.)

SHIFT **(transformation BY** x_shift, y_shift, z_shift**)**
Return a transformation value resulting from shifting the position of the transformation parameter by the given shift amounts.

SIG **(signal_num**, ..., signal_num**)**
Returns the logical AND of the states of the indicated digital signals.

SIG.INS **(signal_num)**
Return an indication of whether or not a digital I/O signal is installed in the system, or whether or not a software signal is available in the system.

SIGN **(value)**
Return the value 1, with the sign of the value parameter.

SIGNAL **signal_num**, ..., signal_num
Turn on or off external digital output signals or internal software signals.

SIN **(value)**
Return the trigonometric sine of a given angle.

SINGLE ALWAYS
Limit rotations of the robot wrist joint to the range –180 degrees to +180 degrees.

SOLVE.ANGLES **o.jts[**o.idx**], o.flags, error = trans, i.jts[**i.idx**],i.flags**
Compute the robot joint positions (for the current robot) that are equivalent to a specified transformation.

SOLVE.FLAGS **(joints[**index**])**
Return bit flags representing the robot configuration specified by an array of joint positions.

SOLVE.TRANS **transform, error = joints[**index**]**
Compute the transformation equivalent to a given set of joint positions for the current robot.

SPEED **speed_factor**, r_speed_factor units ALWAYS
Set the nominal speed for subsequent robot motions.

SPEED **(select)**
Return one of the system motion speed factors.

SPIN speeds[index]
Rotate one or more joints of the selected robot at a specified speed.

SQR **(value)**
Return the square of the parameter.

SQRT **(value)**
Return the square root of the parameter.

STATE **(select)**
Return a value that provides information about the robot system state.

STATUS **(program_name)**
Return status information for an application program.

STOP
Terminate execution of the current program cycle.

STRDIF **($a, $b)**
Compare two strings byte by byte for the purpose of sorting.

SWITCH **switch_name = value**
SWITCH **switch_name[index] = value**
Enable or disable a system switch based on a value.

SWITCH **(switch_name)**
SWITCH **(switch_name[index])**
Return an indication of the setting of a system switch.

$SYMBOL **(pointer)**
Determine the user symbol that is referenced by a pointer previously obtained with the SYMBOL.PTR real-valued function.

SYMBOL.PTR **(string, type)**
Determine the value of a pointer to a user symbol in V+ memory.

TAS **(variable, new_value)**
Return the current value of a real-valued variable and assign it a new value. The two actions are done "indivisibly" so that no other program task can modify the variable at the same time.

TASK **(select, task_num)**
Return information about a program execution task.

TERMINAL
Determine how V+ will interact with the system terminal.

TIME **time_string**
Set the date and time.

TIME **(string, select)**
Return an integer value representing either the date or the time specified in the given string parameter.

$TIME **(date, time)**
Return a string value containing either the current system date and time or the specified date and time.

TIMER **timer_number = time_value**
Set the specified system timer to the given time value.

TIMER **(timer_number)**
Return the current time value (in seconds) of the specified system timer.

TOOL **transformation_value**
Set the internal transformation used to represent the location and orientation of the tool tip relative to the tool mounting flange of the robot.

TOOL
Return the value of the transformation specified in the last TOOL command or instruction.

TPS
Return the number of ticks of the system clock that occur per second (Ticks Per Second).

TRACE
Control the display of program steps on the system terminal during program execution.

TRANS **(X_value, Y_value, Z_value, y_value, p_value, r_value)**
Return a transformation value computed from the given X, Y, Z position displacements and y, p, r orientation rotations.

TRANSB **(string, first_char)**
Return a transformation value represented by a 48-byte string.

$TRANSB **(transformation)**
Return a -byte string containing the binary representation of a transformation value.

TRUE **...**
Return the value used by V+ to represent a logical true result.

$TRUNCATE **(string)**
Return all characters in the input string until an ASCII NUL (or the end of the string) is encountered.

TYPE **output_specification, ..., output_specification**
Display the information described by the output specifications on the system terminal. A blank line is output if no argument is provided.

UNIDIRECT **directions[**index**]**
Specify that a joint is turning only in a single direction.

$UNPACK **(string_array[**index**],**
first_char, num_chars)
Return a substring from an array of 128-character string variables.

UNTIL **expression**
Indicate the end of a DO ... UNTIL control structure and specify the expression that is evaluated to determine when to exit the loop. The loop continues to be executed until the expression value is nonzero.

UPPER
Control whether or not the case of each character is ignored when string comparisons are performed.

VAL **(string)**
Return the real value represented by the characters in the input string.

VALUE **expression_list:**
Indicate the values that a CASE statement expression must match in order for the program statements immediately following to be executed.

WAIT condition
Put the program into a "wait loop" until the condition is TRUE.

WAIT.EVENT mask, timeout
Suspend program execution until a specified event has occurred, or until a specified amount of time has elapsed.

WHILE **condition** DO
Initiate processing of a WHILE structure if the condition is TRUE or skipping of the WHILE structure if the condition is initially FALSE.

WINDOW **%belt_var = location,**
location, program, priority
Set the boundaries of the operating region of the specified belt variable for conveyor tracking.

WINDOW **(transformation**, time, mode**)**
Return a value that indicates where the location described by the belt-relative transformation value is relative to the predefined boundaries of the working range on a moving conveyor belt.

WRITE **(lun**, record_num, mode**)**
format_list
Write a record to an open file, or to an attached device that is file oriented. For an AdeptNet device, write a string to an attached and open TCP connection.

XOR **value ...**
Perform the logical exclusive-OR operation on two values.

System Messages B

Introduction

While the V+ system is being used, it is possible for hardware and software errors to occur. For example, if commands or instructions are not entered in the correct way, V+ rejects the input. The usual response is to write an error message to the system terminal indicating what is wrong so that the user can correct the error.

Alphabetical Listing

The following section contains all the error messages produced by V+, explains what they mean, and indicates what should be done in response. This list also includes a variety of informational messages that V+ displays under certain circumstances.

Almost every V+ message has a numeric code that can be used to identify the message within a V+ program. The ERROR and IOSTAT functions return this error code. The $ERROR string function returns the error message corresponding to an error code. The error code for each message appears at the right margin for all those messages that have a code. For convenience, the second section in this appendix lists all the V+ errors in order of their numerical codes.

All the V+ messages are described in this section. Each description includes the text of the message, its error code, an explanation of the likely cause of the message, and a suggestion of what action you should take.

> **NOTE:** If the system has more than one robot connected and an error is associated with a specific one of the robots, the robot number is appended to the error message in the form (Robot #).

Aborted **(–400)**

Explanation: The last command requested, or the program that was executing, has been aborted at the operator's request.

User action: None.

Already attached to logical unit **(–515)**

Explanation: A program has executed more than one ATTACH instruction for a specific logical unit, without executing a DETACH in between. (The program is still attached to the logical unit after this error occurs.)

User action: Check the program logic and remove redundant ATTACH instructions.

Ambiguous AUTO invalid **(–477)**

Explanation: When exiting from the program editor, V^+ has encountered an automatic variable with undetermined type. That is, the system cannot determine if the variable is real-valued or a transformation. Automatic variables cannot be ambiguous, since their storage requirements must be known before they are referenced.

User action: Include the REAL or LOC type specification parameter in the AUTO statement that declares the variable, or reference the variable in a program instruction in a manner that makes its type clear.

Ambiguous name **(–453)**

Explanation: The abbreviation used for the last command, instruction, or system-defined name was not long enough to identify the operation intended.

User action: Reenter the last line, using a longer abbreviation.

AOI not defined **(–752)**

Explanation: An attempt has been made to place a vision tool using an AOI (area-of-interest) that has not been defined.

User action: Define the AOI using the VDEF.AOI instruction.

Are you sure (Y/N)? **(10)**

Explanation: The requested command will have a significant effect on the state of the system, and V^+ wants to know if you really want it to happen.

User action: To have V+ continue, type **y** followed by a carriage return. An **n** followed by a carriage return or just a carriage return causes the command to be aborted.

***Arithmetic overflow* (–409)**

Explanation: The result of a calculation was outside the allowable range for real variables or V+ has encountered a number that is outside the allowed range for integers while converting a real-valued number to a decimal, hexadecimal, or octal integer, or logical value. Logical values use 32-bit integers, but most program instructions that require integer arguments allow only 16-bit integers. Also, real variables can have only magnitudes in the range from about 5.4E-20 to 9.2E+18.

User action: Modify the program as required.

***A scratch frame store is needed (use VSELECT)* (–756)**

Explanation: VCORRELATE returns this error when performing a grayscale hierarchical search or binary search and no scratch frame store is available.

User action: Use VSELECT to invalidate a virtual frame store in a physical frame store that is different than the physical frame store being searched.

***Attempt to modify active belt* (–614)**

Explanation: A program instruction has been executed that will modify the belt variable that is currently being tracked by the robot.

User action: Change the program in order not to modify the variable while the robot is tracking it.

***Attempt to redefine variable class* variable_name (–470)**

Explanation: Upon exiting from the editor, the named variable was found in two of the following places: the .PROGRAM argument list, an AUTO statement, a LOCAL statement, or a GLOBAL statement.

User action: Modify the program to include the variable in only one of these places.

***Attempt to redefine variable type* variable_name (–469)**

Explanation: If a program is being edited, the line just entered contains a reference to a variable in a manner inconsistent with its use elsewhere in

the program. The most likely problem is confusing a location variable with a real variable. If you just exited from the editor, the named variable conflicts with a global variable that already exists.

User action: If the new use of the variable is correct, you must delete all references to the incorrect variable and then reenter the statement that caused the error. If the new use is incorrect, use a different variable name. If there is a conflict with a global variable, either use a DELETE_ command to delete that variable, or make the conflicting variable AUTO or LOCAL to the current program.

Auto Startup... (None)

Explanation: The automatic start-up procedure has begun. (See the discussion of command programs for more information.)

User action: None required for this message, but subsequent commands in the auto-startup command program may require user action.

WARNING: The robot may begin to move during the automatic start-up procedure. If necessary, you can stop the robot by pressing EMERGENCY STOP on the controller or PANIC on the manual control pendant.

***Bad block in disk header* (–523)**

Explanation: While formatting a disk, a bad disk block has been found in the disk header area. The format operation has failed, and the disk is not usable.

User action: Enter the FORMAT command again—use a different diskette if the error persists.

***Bad camera calibration* (–726)**

Explanation: A VPUTCAL instruction has been used to pass vision calibration data to the AdeptVision system, and one or more of the data elements is not valid.

User action: Make sure the program reads the calibration data from a valid data file, or make sure valid values are asserted by the program.

***Bad grip definition* (–721)**

Explanation: The DEFGRIP instruction was performed with incorrect parameters.

User action: Check the DEFGRIP parameter values. The locations specified by the transformations must not be unreasonably far from the prototype, and the widths and heights of the grip rectangles must not be unreasonably large. An unreasonable distance is one greater than a couple of image widths.

Belt not enabled **(–615)**

Explanation: A robot operation that references a moving conveyor belt has been attempted when the conveyor tracking feature is disabled.

User action: Enter an ENABLE BELT command and retry the operation.

Belt servo dead **(–617)**

Explanation: The belt processor isn't responding to commands from V+.

User action: After saving the programs, power down the controller and power it up again. If this error occurs repeatedly, contact Adept Customer Service.

Belt window violation **(–616)**

Explanation: Either a robot motion has been planned that will move the robot outside of the belt window, or the robot has moved outside of the belt window while tracking the belt.

User action: Modify the program so that the robot will not move outside the belt window. Consult the BELT.MODE parameter and the WINDOW instruction for different ways to define the belt window.

***Branch to undefined label* Step** nnn **(–412)**

Explanation: A program instruction references a program label that is not defined in the program. Either the label is missing or was mistyped when defined or in the reference.

User action: Check the label definition and reference.

Breakpoint at (task) program_name, step n **(17)**

Explanation: A breakpoint was encountered before the indicated step. (Any output associated with the breakpoint is displayed after the message shown above.)

User action: Enter a PROCEED (CTRL+P), RETRY, SSTEP (CTRL+Z), or XSTEP (CTRL+X) command to resume program execution.[1] Otherwise, enter any other monitor command.

Breakpoint not allowed here **(–380)**

Explanation: An attempt has been made to set a breakpoint before the first executable statement of a program.

User action: Enter a new BPT command specifying a step after the first executable statement. That is, after the .PROGRAM statement, any AUTO and LOCAL statements, and all comments and blank lines at the start of the program.

Calibration program not loaded **(–425)**

Explanation: A program required for calibration has not been loaded from disk. This error usually occurs if some of the calibration programs have not been loaded into memory, and the CALIBRATE command or instruction is issued with a input mode that does not allow them to be loaded automatically.

User Action: Reissue the CALIBRATE command or instruction with the proper mode. The default mode of zero causes CALIBRATE to automatically load the required programs from disk, perform the calibration, and then delete the programs.

***Calibration sensor failure* Mtr** n **(–1106)**

Explanation: During calibration, the calibration sensor for the indicated motor could not be read correctly. Either the robot is blocked from moving, or a hardware error has occurred.

User action: Retry the CALIBRATE command or instruction after making sure that the robot is not blocked. If the problem persists, contact Adept Customer Service.

Camera already off **(–719)**

Explanation: A VPICTURE operation to turn the camera off has been processed when the camera is already off (line vision only).

[1] The command keys CTRL+P, CTRL+X, and CTRL+Z are accepted only while using the V+ program debugger in its monitor mode.

User action: Modify the program to remove redundant VPICTURE OFF instructions.

***Camera already running* (–714)**

Explanation: A VPICTURE operation to turn the camera on has been processed when the camera is already running (line vision only).

User action: Modify the program to remove redundant VPICTURE ON instructions, or insert a VPICTURE OFF instruction.

***Camera disconnected* (–710)**

Explanation: The vision interface hardware indicates that the camera is not connected.

User action: Check the camera and cabling to make sure they are connected properly. If the problem persists, consult your vision system manual.

***Camera interface board absent* (–722)**

Explanation: The vision interface board is not responding to a command from the vision system.

User action: Make sure that the vision interface board is installed properly. After saving all the programs and prototypes in memory, power down the controller and power it up again. Consult Adept Customer Service if the problem persists.

***Camera not running* (–705)**

Explanation: An attempt has been made to process a vision system operation when the camera is not running (line vision only).

User action: Enter a VPICTURE ON command and retry the vision operation that failed.

***Cancelled* (–358)**

Explanation: An editor, debugger, or pendant operation has been terminated due to operator intervention.

User action: This is usually an informative message to acknowledge the cancellation of the operation.

***Can't access protected or read-only program* (–310)**

Explanation: An attempt has been made to edit a protected or read-only program. These programs cannot be edited.

User action: None.

***Can't ALTER and track belt* (–626)**

Explanation: Either a belt-relative motion was specified while ALTER mode was enabled, or an attempt was made to enable ALTER mode while the selected robot was tracking a belt. Both operations are prohibited because belt-tracking and ALTER mode cannot be performed at the same time.

User action: Either disable ALTER mode or stop tracking the belt.

***Can't change modes while task running* (–361)**

Explanation: A command was issued to change from debug monitor mode to debug editor mode while the program task being debugged was executing. You can change to debug editor mode only when the associated task is stopped.

User action: Stop execution of the program task being debugged, or continue without using debug editor mode.

***Can't create new slide bar* (–557)**

Explanation: An attempt has been made to create a graphic slide bar in the horizontal or vertical scroll bar. Slide bars should be created only in the main window, although they can appear in the title or menu bars.

User action: Modify the arguments for the GSLIDE instruction to have the slide bar created within the window.

***Can't create program in read-only mode* (–364)**

Explanation: An attempt has been made to initiate editing of a program in read-only access mode, but the program does not exist.

User action: If the program name was entered incorrectly, enter the command again with the correct name. Do not select read-only access (with /R) when creating a new program.

***Can't delete .PROGRAM statement* (–350)**

Explanation: An attempt has been made to delete the .PROGRAM statement while editing a program.

User action: To change the .PROGRAM statement, replace it with another .PROGRAM statement. To delete lines at the beginning of the program, move down to line 2 before issuing delete commands.

***Can't execute from SEE program instruction* (–362)**

Explanation: An attempt has been made to use a SEE editor command that cannot be used after the editor has been initiated with the SEE program instruction.

User action: Enter another command or exit the editor and reenter from the V^+ monitor.

***Can't exit while lines attached* (–355)**

Explanation: You attempted to terminate execution of the editor while lines were present in the attach buffer. The attach buffer must be empty before the editor can be exited.

User action: You can use SHIFT+Copy to deposit the contents of the attach buffer into the current program. You can also use ESC+K to delete lines from the attach buffer (99 ESC+K deletes up to 99 lines from the buffer).

***Can't find calibration program file* (–426)**

Explanation: While processing a CALIBRATE command or instruction, the V^+ system could not find the calibration utility program on the file CAL_UTIL.V2.

User action: Restore the missing file from the V^+ distribution disk to the directory \CALIB\ on the hard disk or working system disk.

***Can't go on, use EXECUTE or PRIME* (–313)**

Explanation: An attempt has been made to continue the execution of a program that has completed or stopped because of a HALT instruction. Normally, an error results when a PROCEED, RETRY, or XSTEP command is entered (or the pendant RUN/HOLD button is pressed) after a program has completed all its cycles.

User action: Use the EXECUTE or PRIME command, or the pendant PRIME function, to restart the program from the desired instruction.

***Can't interpret line* (–450)**

Explanation: V+ could not interpret the last command or instruction entered.

User action: Check the spelling and usage, and reenter the line. In the case of an error while loading from the disk, edit the affected programs to correct the indicated lines—they have been converted to bad lines.

***Can't mix MC & program instructions* (–414)**

Explanation: A program instruction has been encountered during processing of a command program, or an MC instruction has been encountered in a normal program.

User action: Edit the command program to use the DO command to include the program instruction, or remove the MC instruction from the normal program.

***Can't open vision window for read/write* (–734)**

Explanation: An attempt has been made to open the vision window in read/write mode, but the vision system is performing some critical processing that precludes it from releasing the window.

User action: Change the program to have it try the FOPEN again later; or specify /WRITEONLY if no reading will be performed.

***Can't start while program running* (–312)**

Explanation: An attempt has been made to start execution of a program from the manual control pendant while a program is already executing as task #0.

User action: Stop the program currently executing and then retry the operation.

***Cartesian control of robot not possible* (–635)**

Explanation: A program has attempted to perform a straight-line motion with a robot that does not support such motions.

User action: Change the program to use joint-interpolated motion.

***Cat3 diagnostic error* Code** n **(–1108)**

Because these message codes are related primarily to hardware, refer to your Robot Instruction Handbook as your primary source of information. If it does not answer your questions, contact Adept Customer Service. The following table summarizes information about the codes.

WARNING: The test procedures for these messages are for skilled or instructed personnel only. Dangerous voltages are present, including those on the Security Panel. Failure to exercise care can result in death or injury.

Table B-1. Cat3 Diagnostic Error Message Codes

Code n	**Explanation**	**User action**
0	ESTOP board hardware not responding, or Parity error.	Check that the AC supply to the Security Panel is on and that the DC power supply is configured correctly.
1	Hardware state 1 error. An error has occurred in the communication or test sequence.	Try again. If the problem persists, it may be caused by a faulty ESTOP board. Make a note of the error message and code number, and contact Adept Customer Service.
2	Hardware state 2 error. An error has occurred in the communication or test sequence.	Try again. If the problem persists, it may be caused by a faulty ESTOP board. Make a note of the error message and code number, and contact Adept Customer Service.
3	Hardware arm power contactor AP1 error.	Consult your Robot Instruction Handbook or contact Adept Customer Service.
4	Hardware arm power contactor AP2 error.	Consult your Robot Instruction Handbook or contact Adept Customer Service.

Table B-1. Cat3 Diagnostic Error Message Codes (Continued)

Code n	**Explanation**	**User action**
5	Hardware cyclic check relay, channel 1 (SR8) error. An error has occurred in the communication or test sequence.	Try again. If the problem persists, it may be caused by a faulty ESTOP board. Consult your Robot Instruction Handbook or contact Adept Customer Service. Make a note of the error message and code number before contacting Adept Customer Service.
6	Hardware cyclic check relay, channel 2 (SR9) error. An error has occurred in the communication or test sequence.	Try again. If the problem persists, it may be caused by a faulty ESTOP board. Consult your Robot Instruction Handbook or contact Adept Customer Service. Make a note of the error message and code number before contacting Adept Customer Service.

*** Cat-3 external E-STOP* Code** n **(–1111)**

Because these message codes are related to hardware, refer to your Robot Instruction Handbook as your primary source of information. If it does not answer your questions, contact Adept Customer Service. The following table summarizes information about the codes.

Table B-2. Cat3 External E-STOP Error Message Codes

Code n	**Explanation**	**User action**
0	Adept E-stop, channel 1 error	Consult your Robot Instruction Handbook.
1	Adept E-stop, channel 2 error	Consult your Robot Instruction Handbook.
2	Customer E-stop, channel 1 error	Consult your Robot Instruction Handbook.
3	Customer E-stop, channel 2 error	Consult your Robot Instruction Handbook.

***Cat3 external sensor fault* Code** n **(–1109)**

Because these message codes are related to hardware, refer to your Robot Instruction Handbook as your primary source of information. If it does not answer your questions, contact Adept Customer Service.

If one of these message codes occurs, stand away from the robot and attempt to enable power again. If the same error code occurs again for no apparent reason, there may be a fault with the sensor. The following table summarizes information about the message codes.

WARNING: The test procedures for these messages are for skilled or instructed personnel only. Dangerous voltages are present, including those on the Security Panel. Failure to exercise care can result in death or injury.

Table B-3. Cat3 External Sensor Fault Error Message Codes

Code n	**Explanation**	**User action**
0	Accelerometer, channel 1 error. The robot (joint 1 or 2) is moving or accelerating too fast, there is a fault with the accelerometer system, or the accelerometer's built-in test function failed.	If the error occurred while a program was moving the robot, try changing the program to move the robot less quickly or with a lower rate of acceleration or deceleration. For faults with cables or sensors, consult your Robot Instruction Handbook or contact Adept Customer Service.
1	Accelerometer, channel 2 error. The robot (joint 1 or 2) is moving or accelerating too fast, there is a fault with the accelerometer system, or the accelerometer's built-in test function failed.	If the error occurred while a program was moving the robot, try changing the program to move the robot less quickly or with a lower rate of acceleration or deceleration. For faults with cables or sensors, consult your Robot Instruction Handbook or contact Adept Customer Service.

Table B-3. Cat3 External Sensor Fault Error Message Codes (Continued)

Code n	**Explanation**	**User action**
2	Amplifier 3 voltage restrict sensor, channel 1 error. The robot (joint 3) is moving or accelerating too fast, there is a fault with the voltage restrict sensor, or the voltage restrict sensor's built-in test function failed.	If the error occurred while a program was moving the robot, try changing the program to move the robot less quickly or with a lower rate of acceleration or deceleration. For hardware faults, consult your Robot Instruction Handbook or contact Adept Customer Service.
3	Amplifier 3 voltage restrict sensor, channel 2 error. The robot (joint 3) is moving or accelerating too fast, there is a fault with the voltage restrict sensor, or the voltage restrict sensor's built-in test function failed.	If the error occurred while a program was moving the robot, try changing the program to move the robot less quickly or with a lower rate of acceleration or deceleration. For hardware faults, consult your Robot Instruction Handbook or contact Adept Customer Service.
4	Amplifier 4 voltage restrict sensor, channel 1 error. The robot (joint 4) is moving or accelerating too fast, there is a fault with the voltage restrict sensor, or the voltage restrict sensor's built-in test function failed.	If the error occurred while a program was moving the robot, try changing the program to move the robot less quickly or with a lower rate of acceleration or deceleration. For hardware faults, consult your Robot Instruction Handbook or contact Adept Customer Service.
5	Amplifier 4 voltage restrict sensor, channel 2 error. The robot (joint 4) is moving or accelerating too fast, there is a fault with the voltage restrict sensor, or the voltage restrict sensor's built-in test function failed.	If the error occurred while a program was moving the robot, try changing the program to move the robot less quickly or with a lower rate of acceleration or deceleration. For hardware faults, consult your Robot Instruction Handbook or contact Adept Customer Service.

Table B-3. Cat3 External Sensor Fault Error Message Codes (Continued)

Code n	**Explanation**	**User action**
6	Total E-stop, channel 1 (SR5) error	Consult your Robot Instruction Handbook or contact Adept Customer Service.
7	Total E-stop, channel 2 (SR4) error	Consult your Robot Instruction Handbook or contact Adept Customer Service.

Change? **(11)**

Explanation: You are being given an opportunity to modify the location value just created by a HERE or POINT command.

User action: Enter any desired new components, separated by commas, or press the RETURN key to indicate that no changes are desired.

..., change to: **(None)**

Explanation: While initiating a string replacement operation, the SEE editor is prompting for the string to be used for the replacement.

User action: Enter the desired replacement string. Note that if you just press RETURN, the string to be searched for will be erased (that is, an empty string will be used for the replacement).

Character not in font **(–742)**

Explanation: In a string of characters to be recognized by, or trained for, optical character recognition (OCR), one or more characters are not in the current font definition.

User action: Redefine the font to include the missing character(s).

Collision avoidance dead-lock **(–647)**

Explanation: Two robots with collision detection enabled are simultaneously blocking each other's path. That is, neither robot can perform its next motion until the other robot moves out of the way.

User action: Change the application program to prevent the deadlock situation.

Command? **(None)**

Explanation: A SEE editor extended command has been initiated with the X command.

User action: Enter the desired extended command, or press RETURN to cancel the request.

Communication time-out **(–531)**

Explanation: An I/O operation has not completed within the allotted time interval. For data communications, the remote communications device has not properly acknowledged data that was sent.

User action: Make sure the remote device is communicating. Make sure connections to the remote device are operating properly.

Communications overrun **(–524)**

Explanation: Data has been received on an I/O device faster than V^+ is processing it, and some data has been lost. This will happen only on the serial interface line or the network.

User action: Modify the program to service the I/O device more often, add a handshaking protocol, or slow down the transmission rate to V^+.

COMP mode disabled **(–603)**

Explanation: The command attempted requires computer control of the robot, but COMPUTER mode was not selected on the pendant.

User action: Select COMP mode on the pendant or enable DRY.RUN mode from the terminal, then reissue the command.

Controller overheating **(–631)**

Explanation: The temperature sensor in the controller power supply has detected an overheating condition. High power is switched off.

User action: Make sure the controller fans are operating and are not obstructed. Make sure the fan filters are clean. Power down the controller to let it cool off.

Control structure error **(–473)**

Explanation: An incomplete control structure has been encountered during program execution.

User action: Edit the program to correct the control structure.

***Control structure error * Step nn (–472)**

Explanation: V+ has detected an incomplete or inconsistent control structure at the specified step when exiting the program editor, loading a program, or processing a BPT command.

User action: Edit the program to correct the control structure. (Note that the actual error may not be at the indicated step.) If the error occurs in response to a BPT command, you can type **dir /?** to identify programs that are not executable and thus might contain the control-structure error.

***Correlation template too big* (–754)**

Explanation: A vision correlation template has been defined that is too large.

User action: Redefine a smaller template

Cursor at column n (None)

Explanation: The SEE editor WHERE extended command is reporting the current column position of the cursor.

User action: None. This is an informational message.

***Database manager internal error* (–859)**

Explanation: This error indicates that the system has encountered an inconsistency.

User action: Contact Adept Application Engineering. Please record the details of exactly what you were doing at the time the error occurred.

***Data checksum error* (–510)**

Explanation: An error was detected while transferring information to or from an external device.

User action: Attempt the transfer again. If the problem persists, contact Adept Customer Service.

***Data error on device* (–522)**

Explanation: An error was detected while attempting to read information from an external device, possibly because a diskette has been damaged or was not formatted properly.

User action: Attempt the read again. Make sure the correct diskette is being used, that it is properly installed in the drive, and that it is formatted. (Recall that formatting a diskette erases its contents.)

***Data overflow* (–755)**

Explanation: The vision binary correlation hardware has found more matches within the search area than it can process.

User Action: Search a smaller area or redefine your binary template so that it contains more distinguishing features.

***Device error* (–660)**

Explanation: An error was detected for an external device such as one specified in the last DEVICE or SETDEVICE program instruction. The actual error depends upon the type of device referenced.

User action: Check the instruction to make sure the parameters are valid. Refer to the documentation for the device type referenced for information on how to determine what has caused the error.

***Device full* (–503)**

Explanation: There is no more space available on a device. If received for a disk file, the disk is full (if there are many small files on the device, this error indicates the disk directory is full). If received for a servo device, an attempt has been made to assign too many servo axes to a single CPU.

User action: Delete unneeded disk files, or use another drive or diskette. Reconfigure your system so the maximum number of axes per CPU is not exceeded.

***Device hardware not present* (–658)**

Explanation: An attempt has been made to reference a device that is not present in your system.

User action: Check that the device was correctly specified. Check that the device hardware is present and is configured properly.

***Device in use* (–668)**

Explanation: An attempt has been made to attach, assign, or configure a hardware device (e.g., a VMI axis) which is already being used.

User action: Check the program code to make sure the requested device has not already been attached.

***Device not ready* (–508)**

Explanation: (1) The requested disk device (or remote network task) is not prepared to communicate with the V+ system.

(2) A limited-access device like the terminal, the manual control pendant, or a serial line is attached to a different program task.

(3) You have tried to write into a pull-down window while it is displayed.

User action: (1) If the intended device is a system microfloppy disk drive, make sure the diskette is correctly inserted and formatted.

(2) If a limited-access device is specified, ABORT and KILL the program task that has it attached, or wait for the program task to release the device. If the intended device is on the network, check that the proper connections are made and that the remote system is operating correctly. (2) ABORT and KILL the program task that has the device attached, or wait for the task to release the device.

(3) The pull-down menu should not be modified with the FSET instruction while it is being displayed. A suitable time for modifying the pull-down menu is immediately after receiving a menu-selection event.

***Device reset* (–663)**

Explanation: The device is busy processing a reset operation. The reset could have been requested (with a SETDEVICE instruction) by another program task that is accessing the device, or the device could have initiated the reset on its own.

User action: Use software interlocks to prevent a second program task from accessing the device after a reset operation has been requested. (Note that the requesting SETDEVICE instruction will wait for the reset to complete.) Refer to the documentation for the specific device for information on its self-generated resets.

***Device sensor error* (–662)**

Explanation: A hardware error occurred in the sensing system accessed with the last DEVICE instruction.

User action: Refer to the documentation for the sensing system for information on how to determine the cause of the error.

***Device time-out* (–659)**

Explanation: The device has not responded within the expected time.

User action: Check the documentation for the device type referenced for how to determine what has caused the error. Check that the device hardware is configured properly.

***Directory error* (–509)**

Explanation: An error occurred while accessing a disk directory, possibly because the diskette was not formatted or the diskette has been damaged in some way.

User action: Make sure the correct diskette is being used, that it is properly installed in the drive, and that it is formatted. (Recall that formatting a diskette erases its contents.)

***Directory not empty* (–571)**

Explanation: The operation attempted to remove an NFS directory that was not empty.

User action: Delete the directory's contents before deleting the directory.

***DO not primed* (–302)**

Explanation: A DO command was attempted without specifying a program instruction to be executed and no previous DO had been entered.

User action: Provide the desired instruction with the DO command.

***Driver internal consistency error* (–519)**

Explanation: An I/O device or servo has responded in an unexpected manner.

User action: Retry the operation that caused the error. If it persists, contact Adept Customer Service.

***Duplicate character in font* (–740)**

Explanation: A character appears more than once in the string that defines a font for optical character recognition (OCR).

User action: Delete all but one occurrence of each character in the string of characters being defined.

Duplicate model name **(–760)**

Explanation: You are attempting to name a new model with a name that already exists.

User action: Either use a different model name or delete the model with the name you want to use.

Duplicate .PROGRAM arguments **(–468)**

Explanation: At least two of the arguments in a .PROGRAM statement have the same name.

User action: Edit the .PROGRAM line so all the arguments have unique names. (With the V+ SEE editor, you can press the Undo (F6) function key or press ESC+CTRL+C to cancel the changes you have made to a .PROGRAM line.)

Duplicate prototype name **(–718)**

Explanation: The file specified in the current VLOAD command contains a prototype with the same name as one that already exists.

User action: VDELETE the conflicting prototype that already exists. As a precaution, save the existing prototypes first with a VSTORE command.

Duplicate statement label **(–464)**

Explanation: The same program statement label is used more than once in a user program.

User action: Change one of the duplicate labels.

***Duty-cycle exceeded* Mtr** n **(–1021)**

Explanation: The indicated motor has been driven fast for too long a period of time. The servo system has disabled Arm Power to protect the robot hardware.

User action: Turn on Arm Power; reduce the speed and/or acceleration for the motion that was in progress or for motions that preceded that motion; and repeat the motion that failed.

Encoder fault **(–1025)**

Explanation: The servo board has detected a broken encoder wire on the indicated axis.

User Action: Inspect the encoder wiring for intermittent connections or broken wires. Try swapping the encoder cable with another. You can disable this error with the SPEC utility, but do so only as a last resort.

***Encoder quadrature error* Belt** n **(–1013)**

Explanation: The position encoder signal from the specified conveyor belt is sending information that is not phased correctly. The encoder or its cabling may be defective. (Encoder error checking is initiated by the DEFBELT instruction and by enabling the BELT switch while a belt is defined.)

User action: Make sure the encoder cable is properly connected. Try to run the conveyor at a slower speed. Contact Adept Customer Service if the error persists.

***Encoder quadrature error* Mtr** n **(–1008)**

Explanation: The position encoder signal from the specified motor is sending information that is not phased correctly. The encoder or its cabling may be defective.

User action: Turn on high power, calibrate the robot, and try to perform the motion at a slower speed. If the error persists, contact Adept Customer Service.

Enter new value: **(None)**

Explanation: The SEE editor's TEACH command is requesting a new value to be assigned to the selected variable, that is, the one last displayed in the debug window.

User action: Enter the desired new value (as a valid expression for the type of variable selected), or press RETURN to cancel the request.

***Envelope error* Mtr** n **(–1006)**

Explanation: The indicated motor was not tracking the commanded position with sufficient accuracy, indicating a failure in the hardware servo system or something impeding the path of the robot.

User action: Turn on high power and try to perform the motion at a slower speed. Make sure nothing is obstructing the robot motion. If the error recurs, contact Adept Customer Service.

***E-STOP from amplifier* (–641)**

Explanation: The motion interface board has detected an E-STOP condition generated by the motor amplifiers. It indicates that the amplifiers have detected some fault condition.

User action: Check for a subsequent message. To determine if there was an unreported RSC error, type **listr error**(task**,4)**, where task is the number of the task that received the error. If no additional information is available, check that the amplifiers are plugged into the backplane correctly, the fixing screws are tightened, and the motor and signal cables are connected correctly.

***E-STOP from backplane* (–643)**

Explanation: The motion interface board has detected an E-STOP due to the BRAKE-ESTOP signal being asserted on the VMEbus.

User action: Check for a subsequent message. To determine if there was an unreported RSC error, type **listr error(**task**,4)**, where task is the number of the task that received the error. If no additional information is available, call Adept Customer Service.

***E-STOP from robot* (–640)**

Explanation: The motion interface board has detected an E-STOP condition generated by the RSC in the robot. This error is probably due to low air pressure, joint-1 overtravel, or motor overheating. A subsequent error message may provide more information.

User action: Check for a subsequent message.To determine if there was an unreported RSC error,type **listr error(**task**,4)**, where task is the number of the task that received the error. If no additional information is available, check for low air pressure, joint 1 overtravel, or motor overheating.

***E-STOP from SYSFAIL* (–642)**

Explanation: The motion interface board has detected an E-STOP due to the SYSFAIL signal being asserted on the VMEbus.

User action: Check for a subsequent message. To determine if there was an unreported RSC error, type **listr error(**task**,4)**, where task is the number

of the task that received the error. If no additional information is available, call Adept Customer Service.

Executing in DRY.RUN mode (50)

Explanation: The DRY.RUN switch is enabled and program execution has been requested. Thus, no motion of the robot will occur.

User action: None unless motion of the robot is desired. In that case, abort execution of the program and disable the DRY.RUN switch.

***Expected character(s) not found* (–745)**

Explanation: While training characters for subsequent optical character recognition (OCR), the number of characters in the given string did not correspond to the number of characters found in the training window. Character training has been aborted.

User action: Make sure the given string matches the characters in the training window.

***External E-STOP* (–608)**

Explanation: The hardware panic button on the controller or pendant has been pressed, or the external panic circuit has been interrupted, causing high power to be turned off. This message is also displayed if the MANUAL button is pressed or the PANIC command is entered while a robot control program is executing.

User action: If high power is off, release the panic button or restore the external panic circuit. Then turn on high power. If high power is not off, reselect COMP mode on the manual control pendant. Then resume program execution.

***[Fatal] Addr Err* at aaaaaa m:n I=xxxx, A=aaaa, F=ff (None)**

Explanation: An internal problem has occurred with the V$^+$ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart V$^+$ temporarily by pressing CTRL+G. The robot servos will not function, but

you can STORE the programs. Then power down the controller and restart the system.

***[Fatal] Bus Err* at aaaaaa m:n I=xxxx, t=aaaa, F=ff (None)**

Explanation: A computer error occurred because of a bad read from memory, because of noise on the internal data bus, or because of a hardware problem.

User action: To save programs that are in memory, you can restart V+ temporarily by pressing CTRL+G. The robot servos will not function, but you can STORE the programs. Then power down the controller and restart the system. If the problem persists, contact Adept Customer Service.

***[Fatal] CHK Trap* at aaaaaa m:n (None)**

Explanation: An internal problem has occurred with the V+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart V+ temporarily by pressing CTRL+G. The robot servos will not function, but you can STORE the programs. Then power down the controller and restart the system.

***[Fatal] Emul 1010 Trap* at aaaaaa m:n (None)**

Explanation: An internal problem has occurred with the V+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart V+ temporarily by pressing CTRL+G. The robot servos will not function, but you can STORE the programs. Then power down the controller and restart the system.

***[Fatal] Emul 1111 Trap* at aaaaaa m:n (None)**

Explanation: An internal problem has occurred with the V+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart V+ temporarily by pressing CTRL+G. The robot servos will not function, but you can STORE the programs. Then power down the controller and restart the system.

***[Fatal] Illeg Instr* at aaaaaa m:n (None)**

Explanation: An internal problem has occurred with the V+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart V+ temporarily by pressing CTRL+G. The robot servos will not function, but you can STORE the programs. Then power down the controller and restart the system.

***[Fatal] OVF Trap* at aaaaaa m:n (None)**

Explanation: An internal problem has occurred with the V+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart V+ temporarily by pressing CTRL+G. The robot servos will not function, but you can STORE the programs. Then power down the controller and restart the system.

***[Fatal] Priv Viol* at aaaaaa m:n (None)**

Explanation: An internal problem has occurred with the V+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart V+ temporarily by pressing CTRL+G. The robot servos will not function, but you can STORE the programs. Then power down the controller and restart the system.

***[Fatal] Spurious Int* at aaaaaa m:n (None)**

Explanation: An internal problem has occurred with the V+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart V+ temporarily by pressing CTRL+G. The robot servos will not function, but you can STORE the programs. Then power down the controller and restart the system.

***[Fatal] Uninit Trap* at aaaaaa m:n (None)**

Explanation: An internal problem has occurred with the V+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart V+ temporarily by pressing CTRL+G. The robot servos will not function, but you can STORE the programs. Then power down the controller and restart the system.

***[Fatal] ZDIV Trap* at aaaaaa m:n** **(None)**

Explanation: An internal problem has occurred with the V+ software or with the system hardware.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

To save programs that are in memory, you can restart V+ temporarily by pressing CTRL+G. The robot servos will not function, but you can STORE the programs. Then power down the controller and restart the system.

***[Fatal] DIV Instr Err* at aaaaaa m:n** **(None)**

Explanation: The V+ system has detected an error from a divide instruction. This indicates a processor fault.

User action: Power down the controller and try starting it again. If the problem persists, contact Adept Customer Service.

***[Fatal Force Err] ...* ...** **(None)**

Explanation: The force processor has detected an error condition. You can continue to use the V+ system, but the force-sensing system cannot be used until you act upon the error.

User action: None required if you do not intend to use the force-sensing system. Otherwise, refer to the documentation for the force-sensing system for information on how to respond to the error.

[Fatal] Graphics/display processor error **(None)**

Explanation: The graphics processing unit on the graphics system processor has failed to respond to commands from the V+ system.

User action: Power down the controller and try starting it again. If the problem persists, contact Adept Customer Service.

***[Fatal] Initialization failure* Mtr** n **(–1014)**

Explanation: During initialization of a robot kinematic module, the indicated motor failed initialization. The problem may be a missing or improperly configured servo interface board, or an incorrect motor mapping for this module.

User action: Verify that all servo interface boards are correctly installed and configured (use the SPEC.V2 utility for motor mapping). If the problem persists, contact Adept Customer Service.

***[Fatal] Invalid serial I/O configuration* (None)**

Explanation: During initial start-up, V+ has detected that the configuration of the hardware for serial communications is not valid. An attempt has been made to configure a serial unit that is not installed, or an invalid byte format or baud rate has been requested.

User action: Power down the controller and try starting it again. Make sure that the boot disk you are using is valid for your controller. Use the CONFIG_C utility program to make sure the serial I/O configuration is correct. If the problem persists, contact Adept Application Engineering.

***[Fatal] Servo process dead* CPU** n **(–1101)**

Explanation: V+ failed to detect proper operation of the servo process on the indicated CPU. V+ will continue to operate, but will not allow high power to be turned on.

User action: Power down the controller and restart. Use the CONFIG_C.V2 utility to verify that a servo process is enabled for this CPU. If the problem persists, contact Adept Customer Service.

***[Fatal] Servo code incompatible* CPU** n **(–1102)**

Explanation: During initialization, V+ detected an improper version of servo software on the indicated CPU. V+ will continue to operate, but will not allow high power to be turned on.

User action: Power down the controller and restart. If the problem persists, contact Adept Customer Service.

***[Fatal] Servo dead* Mtr** n **(–1104)**

Explanation: The servo process for the indicated motor is not responding to commands from V+. V+ will continue to operate, but will not allow high power to be turned on.

User action: Power down the controller and restart. If the problem persists, contact Adept Customer Service.

***[Fatal] Servo init failure* Board** n **(–1107)**

Explanation: During system initialization the indicated servo interface board could not be initialized. The problem may be an invalid servo configuration, a missing or improperly configured servo interface board, or a hardware failure.

User action: Power down the controller and restart, making sure you are using the correct system disk. If the problem persists, contact Adept Customer Service.

***[Fatal] Stk Overflow* at aaaaaa m:n (None)**

Explanation: A storage stack within V+ has overflowed. If n is 1, the error indicates the V+ monitor has encountered an expression that has parentheses nested too deeply. Any of the following values for n indicates that the program task shown has attempted to evaluate an expression that is too complex to fit in the stack for that task. The value is a hexadecimal number where ^H1 = monitor task and ^HD = task 0, ^HE = task 1,...^H27 = task 26, and ^H28 = task 27.

User action: If the n value is one of those listed above, reduce the complexity of the offending expression. If the value is not one of those listed, an internal problem with V+ is indicated. In that case, it would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and what you were doing at the time the error occurred.

To save programs that are in memory, you can restart V+ temporarily by pressing CTRL+G. The robot servos will not function, but you can STORE the programs. Then power down the controller and restart the system.

***[Fatal] System clock dead* (None)**

Explanation: During initial startup, V+ has failed to detect proper operation of the system clock and timer hardware. V+ cannot run without the clock operating properly.

User action: Power down the controller and try starting it again. If the problem persists, contact Adept Customer Service.

***[Fatal] System clock too fast* (None)**

Explanation: During initial startup, V+ has detected that the system hardware clock is running too fast. V+ cannot run without the clock operating properly.

User action: Power down the controller and try starting it again. If the problem persists, contact Adept Customer Service.

***File already exists* (–500)**

Explanation: There is already a disk file or a graphics window with the name supplied to the last storage request.

User action: Reissue the storage request with a different file name, or delete the old file.

***File already open* (–506)**

Explanation: A disk file or graphics window is already open on a logical unit, and another open request has been attempted.

User action: Modify the program to use a different logical unit number for the file or window you want to open, or perform an FCLOSE operation on the file or window currently open on the specified logical unit number before performing the FOPEN operation.

***File format error* (–512)**

Explanation: The requested disk file is not in a format acceptable to V+ because either it was not created by V+ or the file has been corrupted.

User action: Use another diskette or reference another file.

***File name too long* (–570)**

Explanation: The file name in an NFS operation was too long.

User action: Use a shorter file name.

***File not opened* (–513)**

Explanation: A program request was made to read or write data from a disk device when no file was open, or an attempt was made to access a graphics window that is not open.

User action: Modify the program to open the file or graphics window before attempting to read or write data.

***File or subdirectory name error* (–514)**

Explanation: The specified file name or subdirectory was not a valid disk file name, the directory path contained invalid syntax, or the directory path was too long.

User action: Retry the operation with a correct file name or subdirectory name. Verify that syntax of the directory path is correct. Check that any default directory path specified by the DEFAULT command is correct. Check that the total directory path is not too long when the default is combined with the current file specification.

***File too large* (–569)**

Explanation: The NFS operation caused a file to grow beyond the server's limit.

User action: Close the file, open a new file, and retry the previous operation.

Find: (None)

Explanation: While initiating a string search or replacement operation, the SEE editor is prompting for the string to be found in the program.

User action: Enter the desired search string, or press RETURN to cancel the request.

***First statement must be .PROGRAM* (–351)**

Explanation: An attempt was made to insert or deposit a program statement above the .PROGRAM statement, which must be the first statement in the program.

User action: Move the cursor to below the .PROGRAM line of the program before attempting to insert or deposit statements.

***Font already defined* (–737)**

Explanation: An attempt has been made to VLOAD a font file (for subsequent optical character recognition) that contains a font with the same number as one already in memory. The load operation has been aborted and none of the fonts in the file have been loaded.

User action: Rename or delete the font currently defined in memory.

***Font not completely trained* (–738)**

Explanation: During planning of a font for optical character recognition (OCR), some or all of the characters in the font have not been trained.

User action: Display the font (with VSHOW.FONT) to see which characters have not been trained. Then train those characters or delete them from the font.

Font not defined **(–736)**

Explanation: The font specified for optical character recognition (OCR) is not defined.

User action: Use VDEF.FONT or VLOAD to define the font.

Font not loaded **(–551)**

Explanation: The specified font does not exist.

User action: Specify another font (font #1 is always loaded).

Force protect limit exceeded **(–624)**

Explanation: At least one force-sensor strain gauge reading has exceeded the preset limit, causing a robot panic stop. This may happen due to high forces experienced during an insertion, a crash, or high acceleration.

User action: If a crash occurred, ensure that the work area is cleared. If the limit was exceeded in normal operation, the limit should be increased or Protect mode should be disabled. Enable high power with the manual control pendant and continue operation.

Function already enabled **(–422)**

Explanation: Certain functions or operations must not be enabled when they are already enabled or active. ALTER mode is an example of such a function.

User action: Avoid reenabling the function or operation.

Graphics processor timeout **(–552)**

Explanation: The graphics processor (on the system processor) failed to respond to a command from V^+ within five seconds.

User action: Save all your programs and variables on disk and then reboot the system from disk. Contact Adept Customer Service if the problem repeats.

Graphics software checksum error **(–558)**

Explanation: The code on the graphics board has been corrupted.

User action: Save new or modified programs, restart the controller, and reload the programs. If the problem persists, contact Adept customer service.

Graphics storage area format error **(–555)**

Explanation: During execution of a FREE command, V+ has detected that the information in graphic memory may have been corrupted. This may have been caused by a momentary hardware failure or a software error.

User action: Attempt to save as much as possible onto disk. Issue ZERO 1 and ZERO 2 monitor commands to delete graphics data. If the error persists, power down the controller and restart the system.

(HALTED) **(8)**

Explanation: A HALT instruction has been executed, and thus execution of the current program has terminated.

User action: Any monitor command can be entered, but PROCEED cannot be used to resume program execution.

***Hard envelope error* Mtr** n **(–1027)**

Explanation: The indicated motor was not tracking the commanded position with sufficient accuracy, indicating a failure in the hardware servo system or something impeding the path of the robot. Because this is considered a serious error, high power was turned off.

User Action: Turn on high power and try to perform the motion at a slower speed. Make sure that nothing is obstructing the robot's motion. If the error recurs, contact Adept Customer Service.

Hardware not in system **(–805)**

Explanation: An instruction has attempted to access optional hardware (such as a FORCE board) that is not installed in the system.

User Action: Install the needed hardware or remove the instruction that addresses the hardware.

HIGH POWER button on VFP not pressed **(–646)**

Explanation: You did not press the HIGH POWER ON/OFF button on the VFP before the timeout period expired.

User action: If working from the keyboard, reissue the ENABLE POWER monitor command and promptly press the HIGH POWER ON/OFF button when instructed to do so. If working from the MCP, follow the procedure appropriate for enabling high power for the safety cate-

gory of your system. Promptly press the HIGH POWER ON/OFF button when instructed to do so. If the timeout period is too short, adjust it by using the CONFIG_C utility to change the POWER_TIMEOUT statement in the V+ configuration data.

This message also can result from a faulty cable, VFP, or SIO.

Illegal array index **(–404)**

Explanation: An attempt has been made to: (1) use a negative value as an array index, (2) use a value greater than 32767 as an array index, (3) specify a simple variable where an array variable is required, (4) omit an array index in a situation where it is required (for example, a 1-dimension array is specified when a 2- or 3-dimension array is required), (5) specify an explicit index in an argument for a V+ operation that requires a null array, or (6) specify an index to the right of a blank index for a multiple-dimension array.

User action: Correct the line.

Illegal assignment **(–403)**

Explanation: The assignment operation just attempted was invalid, possibly because it attempted to assign a value to a variable name that is a reserved word or a function.

User action: Reenter the line, using a different variable name if necessary.

Illegal camera number **(–803)**

Explanation: A vision command or instruction has specified a camera number value that is invalid.

User action: Reenter the command or edit the program using the correct camera number.

Illegal digital signal **(–405)**

Explanation: A number or bit field specifies a digital signal that is not in one of the allowed ranges or that is not installed. Attempting to set software signal 2032 (brake solenoid) will also give this error.

User action: Correct the signal number and check your digital I/O configuration.

***Illegal expression syntax* (–458)**

Explanation: While decoding a numeric or logical expression, a compound transformation, or a string expression, V+ has encountered syntax that it does not understand. Possible errors include unmatched parentheses, missing variables, or missing operators.

User action: Retype the line containing the expression, being careful to follow the V+ syntax rules.

***Illegal in debug monitor mode* (–359)**

Explanation: An operation was attempted that is not accepted in debug monitor mode.

User action: Use a different command, change to debug editor mode, or exit from the program debugger.

***Illegal in read-write mode* (–365)**

Explanation: An editor function was attempted that cannot be performed while accessing a program in read-write mode.

User action: Change to editing the program in read-only mode, or use a different editor command.

***Illegal I/O channel number* (–518)**

Explanation: An internal I/O channel number has been encountered that is invalid. This indicates a V+ internal software problem.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

***Illegal I/O device command* (–502)**

Explanation: A command to an I/O device was rejected by that device. Certain devices will not accept all commands. For example, random access I/O is illegal to the terminal or to the Kermit device; the GETC function cannot read from a disk file opened for random access. This error may also indicate a hardware problem with the device controller.

User action: Correct the I/O command as required to suit the device. If you continue to have difficulty, contact Adept Application Engineering for assistance.

***Illegal I/O redirection specified* (–525)**

Explanation: An unacceptable I/O redirection has been specified in a DEFAULT monitor command, a disk I/O monitor command (LOAD or STORE_), or in an ATTACH instruction. Either there is a syntax error, or the requested redirection is not allowed for your I/O configuration.

User action: Check the syntax of the offending statement. Check your I/O configuration to make sure the requested redirection device is allowed.

***Illegal joint number* (–609)**

Explanation: A joint number has been specified out of the allowed range.

User action: Correct the joint number.

***Illegal memory reference* (–418)**

Explanation: An operation has attempted to reference an invalid memory address. That is, one that is either out of the allowed range, or that is not in use for any input/output module.

User action: Correct the address or install the missing module.

***Illegal monitor command* (–300)**

Explanation: The name of the command just attempted was not recognized by the system, possibly because it was mistyped or because it was a program instruction and not a command.

User action: Check the spelling of the command name and enter the command again. Use the DO command to invoke a program instruction from the terminal.

***Illegal motion from here* (–613)**

Explanation: The motion just attempted cannot be performed from the current robot location. For example, NEST can be executed only immediately after a READY instruction; CALIBRATE can be executed only after power-up, LIMP, or NEST; and only CALIBRATE or READY can be executed when the robot is in the nest.

User action: Perform the appropriate operation sequence before retrying the desired motion.

***Illegal operation* (–423)**

Explanation: A program instruction has attempted to perform an operation that is not possible.

User action: Check the instruction executing when the error occurred. Make sure all conditions necessary for its successful completion are met.

***Illegal .PROGRAM statement* (–467)**

Explanation: An attempt has been made to: (1) enter a line other than a .PROGRAM statement as the first line of a program, or (2) enter a .PROGRAM statement that contains a syntax error.

User action: Move below the first line of the program, or reenter the line correctly. (With the V$^+$ SEE editor, you can press the Undo function key or press ESC+CTRL+C to cancel the changes you have made to a .PROGRAM line.)

***Illegal record length* (–528)**

Explanation: An FOPEN instruction has specified a record length that is not acceptable. For example, the value is negative or too large, or the record length is zero with random-access mode specified.

User action: Edit the program to specify a correct record length or specify sequential-access mode.

***Illegal use of belt variable* (–466)**

Explanation: A belt variable has been used in a context where it is not allowed, probably in a compound transformation but not at the left-most position.

User action: Edit the program to use the belt variable correctly.

***Illegal user LUN specified* (–527)**

Explanation: An I/O instruction has specified a logical unit number (LUN) that is not defined in the V$^+$ system, or cannot be accessed in the manner attempted. (See the description of the ATTACH instruction for a list of the valid logical unit numbers and the devices to which they apply.)

User action: Edit the program to use a logical unit number appropriate for the instruction.

***Illegal value* (–402)**

Explanation: A numeric or expression value that is not in the allowed range was specified within a command or instruction.

User action: Edit the program to use a legal value.

***Illegal when command program active* (–419)**

Explanation: A command program is active and an attempt has been made to execute a command that would interfere with operation of the command program. (For example, processing a ZERO command would cause the command program to be deleted from the system memory.)

User action: Edit the command program and delete the command causing the error.

***Illegal when network enabled* (–543)**

Explanation: An attempt has been made to perform certain network functions that require that the network be disabled, but the network is enabled.

User action: Disable the network and retry the operation.

***Illegal while joints SPIN'ing* (–637)**

Explanation: An attempt has been made to execute a regular motion instruction while a SPIN trajectory is being executed.

User action: Stop the SPIN trajectory with a SPIN or BRAKE instruction before executing a regular motion instruction.

***Illegal while protocol active* (–548)**

Explanation: This message indicates that the user tried to enter passthru mode, or did something unexpected on the serial line configured for use with Kermit while a file was being processed.

User action: Make sure there is no file being accessed by Kermit, and retry the failed operation.

***Image processing board failure* (–728)**

Explanation: The controller circuit board that processes vision images has failed to respond while processing a request to grab a frame.

User action: After saving the programs, variables, and vision prototypes in memory, power down the controller. Make sure the image processor is firmly seated in the controller backplane. Contact Adept Customer Service if the problem persists.

***Incompatible robot and safety ID* (–644)**

Explanation: The robot and controller do not have the same safety options.

User action: Make sure that the correct robot and controller are being used together. Install (or remove) the appropriate EN954 Safety Category license in the controller.

***Inconsistent hierarchy levels* (–757)**

Explanation: The VPLAN.FINDER vision instruction has attempted to combine two or more object finder models that were not trained at the same hierarchiacal (subsample) level.

User action: Retrain the models so that they are all at the same hierarchiacal levels.

***Information not available* (–730)**

Explanation: (1) A VGETPIC, VPUTPIC, VRULER, VRULERI, or VWINDOW operation has been attempted when the specified frame store (binary or grayscale) does not contain valid picture data. (2) No information is available for VGAPS or VSUBPROTO (for example, V.LAST.VER.DIST is set to zero), or the prototype name specified is not the name of the last object located.

User action: Change the operations that precede the failed one to make sure the required conditions are satisfied.

***Initialization error* (-505)**

Explanation An I/O device reported an error condition during its initialization. Initialization is performed during power-up, after a reset, and may also be performed after certain nonrecoverable I/O errors occur.

User action Be sure that the hardware for the I/O device is properly installed. Repeat the failed I/O operation. If the problem persists, contact Adept Field Service.

***Initialization failure* Belt** n **(–1015)**

Explanation: The indicated belt encoder monitoring system failed to respond to V+ during the initialization caused by the DEFBELT instruction.

User action: Power down the controller and restart. If the problem persists, contact Adept Customer Service. (You can prevent this error from being reported by enabling the DRY.RUN system switch.)

Input block error **(–511)**

Explanation: A read error has occurred while reading a binary data file from the floppy disk. This indicates that the wrong file was specified or that the data in the file is corrupted.

User action: Try the operation again. If the error recurs use another diskette.

***Input error* Try again:** **(16)**

Explanation: The input provided was not consistent with what V+ expected.

User action: Provide another response.

Invalid argument **(–407)**

Explanation: An argument for a function, program instruction, or SEE editor command is not in the accepted range.

User action: Check the range of arguments for the function, program instruction, or editor command being used.

Invalid camera calibration **(–802)**

Explanation: A vision system operation has been attempted before the camera-to-robot calibration has been done.

User action: Execute the camera-to-robot calibration program provided by Adept, or load previous calibration data. The latter can be done, for example, by calling the subroutine load.line provided on the Adept Utility Disk in the file LOADAREA.V2.

Invalid character in font **(–741)**

Explanation: An invalid character appears in the string that defines a font for optical character recognition (OCR). The characters in the string must be in the range ASCII 33 (!) to 126 ().

User action: Delete the invalid character from the string.

Invalid connection specified **(–540)**

Explanation: An invalid logical network connection has been specified. For example, a zero connection ID is invalid.

User action: Specify a valid logical connection ID.

Invalid disk format **(–520)**

Explanation: An attempt has been made to read a disk that is not formatted, or is formatted improperly; or a FORMAT command has been entered that specifies invalid format parameters for the device specified.

User action: If a FORMAT command has been entered, check the command syntax and retry the command. Otherwise, try a different diskette or reformat the current one. Remember that formatting erases all information on the diskette. If the diskette was created on an IBM PC, be sure it was formatted with one of the formats accepted by the V$^+$ system.

***Invalid error code* Belt** n **(–1010)**

Explanation: An unrecognized error was reported by the controller for the indicated conveyor belt.

User action: Attempt the operation again. If the error repeats, report the situation to Adept Application Engineering.

Invalid format specifier **(–461)**

Explanation: An unrecognized output format specifier was encountered in a TYPE or WRITE instruction, or in an $ENCODE function.

User action: Edit the program to use a valid format specifier.

Invalid hardware configuration **(–533)**

Explanation: An attempt has been made to access an I/O device in a manner inconsistent with its current configuration. Either the I/O device is

not present in the system, or it is configured for some other use. For example, if a serial communication line is configured as a network port, it cannot be accessed as a user serial line.

User action: Make sure the correct device is being accessed. Power down the controller and try starting it again. Make sure the boot disk you are using is valid for your controller. Use the CONFIG_C utility program to make sure the serial I/O configuration is correct. If the problem persists, contact Adept Application Engineering for assistance.

If the error resulted from a disk I/O operation, it indicates that the disk controller hardware is not configured correctly. Adept Customer Service should be contacted in that case.

Invalid in read-only mode **(–352)**

Explanation: An editor function was attempted that cannot be performed while accessing a program in read-only mode.

User action: Change to editing the program in read-write mode, or use a different editor command.

Invalid model name (–732)

Explanation: The name of a prototype, subprototype, OCR font, or correlation template has been incorrectly specified. The correct format for prototype names is proto:subproto, where proto is a prototype name and subproto is a subprototype name. This error occurs if the colon is followed by a blank, or when some other character is used instead of a colon. Font names have the form FONT_n, where n is an integer in the range 0 to 50. (The special name FONT_0 refers to all fonts.) Similarly, template names have the form TMPL_n. Prototype names should not begin with FONT_ or TMPL_.

User action: Enter the attempted operation again, correctly specifying the prototype, subprototype, OCR font, or correlation template.

Invalid network address **(–561)**

Explanation: This error occurs when an NFS server has not correctly exported the path being accessed or when an IP network address specified is not of class A, B, or C.

User action: Check the IP addresses used to refer to network nodes.

***Invalid network protocol* (–541)**

Explanation: A message has been received and rejected by a remote node because it does not follow the expected protocol. If the KERMIT device was being accessed, this error indicates the remote server reported an error or sent a message not understood by the V+ Kermit driver.

User action: Check that network software version on the remote node is compatible with the network software on the local node. DISABLE and ENABLE the affected network nodes and retry the operation. If this error occurs repeatedly, contact Adept Application Engineering for assistance. (See the ***V+ Language User's Guide*** for information on Kermit.)

Invalid network resource* (–560)

Explanation: This error occurs when referring to a node that has not been defined.

User action: Check the node definitions.

***Invalid number format* (–456)**

Explanation: A syntax error was detected while reading a number. For example, an 8 or 9 digit was encountered while reading an octal number.

User action: Reenter the line with a valid number.

***Invalid orientation* (–619)**

Explanation: A motion has been requested to a location that is defined by a transformation with its orientation pointed up instead of down.

User action: Correct the definition of the destination transformation. For example, you may need to correct the base transformation in the compound transformation. (The p component of all destination transformations should be approximately 180 degrees.)

***Invalid program or variable name* (–455)**

Explanation: A user-defined name used in a command or instruction was not recognized by V+.

User action: Check the name and retype the line.

Invalid qualifier **(–476)**

Explanation: An invalid qualifier was specified on the last command.

User action: Enter the command again, with a valid qualifier.

Invalid request while camera running **(–706)**

Explanation: An operation was attempted that requires the vision system to be idle, but it was still processing an image.

User action: Use a VWAIT instruction to make program execution wait for the vision system to be idle.

Invalid request while vision training **(–729)**

Explanation: An VEDGE.INFO or VGAPS instruction has been attempted while the vision system is in prototype training mode.

User action: Use the manual control pendant to terminate prototype training, type Ctrl+C at the system terminal to abort a VTRAIN command, or abort execution of the program that initiated training.

***Invalid servo error* Mtr** n **(–1001)**

Explanation: An unrecognized error was reported for the indicated robot motor.

User action: Attempt the operation again. Contact Adept Customer Service if the error repeats.

Invalid servo initialization data **(–625)**

Explanation: During V^+ system initialization after booting from disk, servo initialization data in the wrong format was found. This can be caused by using a version of the SPEC utility that is incompatible with the V^+ system.

User action: Make sure your system disk has been configured correctly. Contact Adept Application Engineering for assistance.

Invalid software configuration **(–315)**

Explanation: During initial startup, V^+ has detected that the system software is not configured properly for the options or hardware present.

User action: Power down the controller and try starting it again. Make sure that the boot disk you are using is valid for your controller. If the problem persists, contact Adept Customer Service for assistance.

Invalid statement label **(–463)**

Explanation: The program statement label was not an integer from 0 to 65535.

User action: Reenter the line with a valid label.

Invalid steps will be changed to ? lines **(None)**

Explanation: The AUTO.BAD extended command has been used to change the action to be taken when an invalid line is detected while editing. Subsequently, such a line will automatically be changed to a bad line with a question mark in column one.

User action: None. This is an informational message.

Invalid VFEATURE access **(–801)**

Explanation: A VFEATURE function has been used to access, from the vision system, data that is not valid. In particular, after a VLOCATE instruction in no-wait mode, the vision data is invalid if VFEATURE(1) returns the value FALSE.

User action: Edit the program to ensure that, after a no-wait VLOCATE, no VFEATURE accesses [other than VFEATURE(1)] occur if the vision data is indicated by VFEATURE(1) to be invalid.

Invalid vision argument **(–735)**

Explanation: An argument for a vision function, program instruction, or command is not in the accepted range.

User action: Check the acceptable range of arguments for the function, program instruction, or command being used. Check the vision calibration to make sure the scaling is reasonable.

Invalid vision X/Y ratio **(–727)**

Explanation: A VPUTCAL instruction has been used to pass vision calibration data to the AdeptVision system, and the x-scale to y-scale is not in the acceptable range.

User action: Make sure the program reads the calibration data from a valid data file, or make sure valid values are asserted by the program.

***Invalid when program on stack* (–366)**

Explanation: An attempt has been made to edit a .PROGRAM or AUTO statement while the program appears on some task execution stack. While a task is on a stack, its subroutine arguments and automatic variable values are kept on the stack. Changes to these statements would modify the stack, which is not allowed.

User action: Remove the program from the stack by allowing the task to run until the desired program executes a RETURN instruction, or issue a KILL monitor command to clear the stack. If you are using the SEE program editor, press the Undo key to allow you to continue editing.

***Invalid when program task active* (–311)**

Explanation: An attempt has been made to begin execution of a robot or PC program task when that task is already active.

User action: Abort the currently executing task, or execute the program as a different task, if possible.

***I/O communication error* (–507)**

Explanation: A hardware error has been detected in the I/O interface.

User action: Try your command again. If the problem persists, contact Adept Customer Service.

***I/O queue full* (–517)**

Explanation: Too many I/O requests have been issued to a device too quickly, and there is no more room to queue them.

User action: Retry the operation. If the problem persists, it would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred.

***Is a directory* (–568)**

Explanation: The caller specified a directory in a nondirectory NFS operation.

User action: Specifying a file that is not a directory, repeat the operation; or perform the correct directory operation.

***Joint 1 in brake track or robot overheated* (–606)**

Explanation: (1) Robot joint 1 has been moved into the hardware brake track area, which causes high power to be turned off and prevents the robot from moving.

(2) The robot base has become overheated.

User action: (1) Push the brake release button at the robot base and move the joints back into the normal working range. Turn on high power and continue program execution.

(2) Check the fan filter on the robot base, and check the ambient temperature of the robot. Allow the robot to cool down, turn on high power, and continue program execution.

***Keyswitch not set to AUTO* (–303)**

Explanation: An attempt has been made to PRIME or otherwise initiate program execution from the terminal when the front-panel keyswitch is not set to the AUTO position.

User action: Move the keyswitch to the AUTO position or start program execution from the selected device.

***Keyswitch not set to MANUAL* (–304)**

Explanation: An attempt has been made to PRIME or otherwise initiate program execution from the manual control pendant, when the front-panel keyswitch is not in the MANUAL position. If you do not have a front panel, the keyswitch is assumed to be set to the AUTO position.

User action: Move the keyswitch to the MANUAL position, or start program execution from the selected device.

***Keyswitch not set to NETWORK* (–317)**

Explanation: An attempt has been made to use a serial line configured for network use, but the front-panel keyswitch is not in the NETWORK position. If you do not have a front panel, the keyswitch is assumed to be set to the AUTO position.

User action: Move the keyswitch to the NETWORK position, and retry the operation.

Line too long **(–354)**

Explanation: An operation was attempted that would have resulted in accessing a program step that contains too many characters. A single program step can contain at most about 150 characters.

User action: Enter the program step as two or more separate steps.

Location out of range **(–610)**

Explanation: V^+ has encountered a location that is too far away to represent (possibly within an intermediate computation) or that is beyond the reach of the robot. This probably indicates an error in a location function argument value or in a compound transformation.

User action: Check to make sure you are using location functions and operations correctly and edit the program as required.

Location too close **(–618)**

Explanation: An attempt has been made to move the robot to a location that is too close to the robot column. This probably indicates an error in the value of a location function argument or an incorrect compound transformation.

User action: Check to make sure you are using location functions and operations correctly and edit the program as required.

Macro (Z ends): **(None)**

Explanation: Definition of a SEE editor macro command has been initiated.

User action: Enter the keystrokes to define the macro and then enter Z to terminate the definition.

Manual brake release **(–639)**

Explanation: The robot's manual brake-release button is active. It is not possible to enable power when this button is pressed.

User Action: Make sure that the manual brake-release button (usually located on the robot) is not active. If the problem persists even though the button is not pressed, call Adept Customer Service.

Manual control pendant failure **(–650)**

Explanation: A program has attempted to access the manual control pendant when it is disconnected or has failed.

User action: Make sure the pendant is connected properly. If the problem persists, contact Adept Customer Service.

Maximum number of prototypes exceeded **(–712)**

Explanation: A maximum of 25 prototypes may be in the AdeptVision system memory at one time.

User action: If not all of the current prototypes are needed, then store them on disk using the VSTORE monitor command and VDELETE the prototypes that are not needed. This will reduce the number of prototypes in memory so that more may be VTRAINed or VLOADed.

Maximum number of samples trained **(–739)**

Explanation: An attempt has been made to train a character more than 30 times for optical character recognition (OCR).

User action: Display the font (with VSHOW.FONT) and determine which characters have already been trained 30 times. Don't train those characters any more.

***Memory Err* at aaaaaa** **(None)**

Explanation: During initialization, V^+ detected a hardware failure at the indicated memory location.

User action: Power down the controller and start it again. If the error persists, contact Adept Customer Service.

Misplaced declaration statement **(–471)**

Explanation: Upon loading a program or exiting from the program editor, V^+ has encountered an AUTO or LOCAL statement that follows an executable program instruction.

User action: Edit the program to make sure that AUTO and LOCAL statements are preceded only by blank lines, comments, or other AUTO and LOCAL statements.

Missing argument **(–454)**

Explanation: A valid argument was not found for one or more of the arguments required for the requested command or instruction. That is, the argument was not present at all or an invalid argument was present. A possible cause is the use of a single equal sign (=) for the equality relational operator (==).

User action: Check the operation syntax and reenter the line.

Missing bracket **(–475)**

Explanation: In the specification of an array element, a left bracket has been found with no matching right bracket. Either too many left brackets are present or a right bracket has been omitted.

User action: Reenter the line with correctly matching left and right brackets.

Missing parenthesis **(–459)**

Explanation: An attempt was made to evaluate an expression that did not have correctly matching left and right parentheses.

User action: Correct the expression.

Missing quote mark **(–460)**

Explanation: A quoted string has been encountered that has no matching quote mark before the end of the line.

User action: Insert a quote mark at the end of the string. Strings may not cross line boundaries.

Mixing half and full resolutions **(–750)**

Explanation: A model (recognition prototype, OCR font, or correlation template) was defined using a full-frame image, but was applied to a half-frame image (field only), or vice versa.

User action: Make sure the correct virtual camera is being used for both defining the model and applying the model. Associated with each virtual camera is a calibration array that contains information indicating whether full-frame or half-frame images are to be acquired with the virtual camera.

Motion interface E-STOP **(–630)**

Explanation: The AdeptMotion system has detected an error or problem and has asserted the BRKSTOP signal on the VMEBus. If that error is seen, it indicates a transient BRAKE-ESTOP signal or a problem with either the motion interface board or the SIO module.

User action: Correct the problem that is causing the motion system to report the error.

***Motor amplifier fault* Mtr** n **(–1018)**

Explanation: The power amplifier for the indicated motor has signaled a fault condition on fault line 1. This fault occurs only for devices controlled by the AdeptMotion Servo system. The interpretation of this fault depends on the particular device being controlled.

User action: Turn high power back on and restart the program. If the error persists, implement procedures appropriate for your AdeptMotion system. If the robot is a standard Adept product, contact Adept Customer Service.

***Motor overheating* Mtr** n **(–1016)**

Explanation: The indicated motor is overheating.

User action: Reduce the speed, acceleration, and/or deceleration of the robot motions, or introduce delays in the application cycle to give the motor an opportunity to cool.

***Motor stalled* Mtr** n **(–1007)**

Explanation: The indicated motor has stalled while being driven. This is usually caused by the robot encountering an obstruction.

User action: Turn high power back on and restart the program. Remove the obstruction or modify the program to have the robot follow a different path.

***Motor startup failure* Mtr** n **(–1105)**

Explanation: During calibration, the indicated motor did not move as expected. The problem may be: (1) the motor is obstructed or up against a limit stop, (2) the load on the robot is too large for calibration, (3) the motor drive hardware is not functioning, or (4) the position encoders are not functioning.

User action: Move the robot away from its limit stops and remove any unusual load. Turn high power back on and try to calibrate again. Contact Adept Customer Service if the error persists.

***Must be in debug mode* (–360)**

Explanation: An editor function was attempted that is accepted only when the program debugger is active.

User action: Use a different editor command or activate the program debugger with the SEE editor DEBUG extended command or the DEBUG monitor command.

***Must use CPU #1* (–666)**

Explanation: A command or instruction that requires execution on CPU #1 has been attempted on a different CPU.

User: action: Reexecute the command or instruction on CPU #1.

***Must use straight-line motion* (–611)**

Explanation: A joint-controlled motion instruction was attempted while the system was in a mode requiring that only straight-line motions be used. For example, while tracking a conveyor, only straight-line motions can be used.

User action: Change the motion instruction to one that requests a straight-line motion.

Negative overtravel* Mtr** n (–1032)**

Explanation: The indicated motor has moved beyond the hardware-limited negative range of motion.

User action: Move the robot back into the working envelope. Correct whatever caused the robot to get into the restricted area. Then enable power.

***Negative square root* (–410)**

Explanation: An attempt has been made to calculate the square root of a negative number.

User action: Correct the program as required.

Network closed locally **(–535)**

Explanation: An attempt has been made to access a DDCMP serial line when the protocol is not active. The protocol was probably stopped because of some other error condition.

User action: Restart the DDCMP protocol.

Network connection closed **(101)**

Explanation: A client connection has closed on the given logical unit.

User action: None. This is an information message.

Network connection opened **(100)**

Explanation: A new client connection has been established on the given logical unit.

User action: None. This is an information message.

Network connection terminated **(–565)**

Explanation: This error occurs when input or output operations are attempted on a network connection that has already been terminated.

User action: Reestablish the network connection, and retry the original operation.

Network error* Code *n **(value received)**

Explanation: An error code between –255 and –1 (inclusive) has been received from the network. The error code, which does not have meaning to V+, is being reported to the user.

User action: Application dependent. If the indicated code does not having meaning for the current application, check to make sure the remote computer is sending valid data.

Network node off line **(–538)**

Explanation: An attempt has been made to send data to a known network node that is off-line. Either the node has been disabled, or it is not connected to the network.

User action: Check that the remote node is active and connected to the network. Check that the local node is connected to the network.

Network not enabled **(–542)**

Explanation: An attempt has been made to access a remote network node, or perform certain network functions, when the network is not enabled.

User action: Enable the network and retry the operation.

Network resource name conflict **(–564)**

Explanation: The name specified for an NFS mount matches an existing network name such as an NFS disk name.

User action: Choose a different name.

Network restarted remotely **(–534)**

Explanation: V+ has received a DDCMP start-up message from the remote system when the protocol was already started. The remote system has probably stopped and restarted its protocol. The local protocol is stopped and all pending I/O requests are completed with this error.

User action: (1) Close and reopen the DDCMP line; (2) check the remote program logic to see why it restarted the protocol.

Network timeout **(–562)**

Explanation: This error occurs when a network transaction is initiated but no reply is received from the server.

User action: Check network integrity. Make sure the server is up and running. Make sure the correct IP address is being used.

NFS error* Code *n **(–1200 to –1299)**

Explanation: Because NFS returns errors that do not have corresponding meaning in V+, some NFS errors are reported as a variable NFS error. Errors in this range have the following interpretation: V+ error number –(1200+n) corresponds to NFS error code n. Following are the currently known NFS error codes that are reported in this way:

Table B-4. NFS Error Message Codes

Code *n*	**Explanation**
19	No such device.
30	Read-only file system. A write operation was attempted on a read-only file system.

Table B-4. NFS Error Message Codes

Code *n*	Explanation
69	Disk quota exceeded. The client's disk quota on the server has been exceeded.
99	The server's write cache used in the WRITECACHE call was flushed to disk.

No air pressure **(–607)**

Explanation: V+ detected that the air supply to the robot brakes and hand has failed. High power is turned off and cannot be turned on until the air pressure is restored.

User action: Restore the air pressure, turn high power back on, and resume program execution. If the error persists, contact Adept Customer Service.

No data received **(–526)**

Explanation: An I/O read request without wait has not found any data to return. This is not really an error condition.

User action: Continue polling the I/O device until data is received, or use a read request that waits automatically for data to be received.

No matching connection **(–539)**

Explanation: A request for a logical network connection has been received and rejected because there is no matching connection on the remote node.

User action: Check that the proper logical connection was specified. Check that the remote node is operating properly.

No models **(–758)**

Explanation: The VSTORE program instruction or monitor command has not found any models to store.

User action: Supply correct model names to the VSTORE instruction.

No models planned **(–761)**

Explanation: Recognition cannot commence because no object models have been planned for the given virtual camera.

User action: Supply the correct virtual camera number or plan that uses the specified camera number. (See the VPLAN.FINDER instruction in the ***AdeptVision Reference Guide***.)

No objects seen **(–704)**

Explanation: The vision system reports that no objects were seen, in response to a VTRAIN or VLOCATE command. In the VLOCATE case, it is an error only if you expect to see objects.

User action: In the training case, make sure the training object is visible under the camera. If you expect to see objects, check the threshold parameter, the minimum area parameter, and the camera hardware.

No other program referenced **(–353)**

Explanation: A command was issued that attempted to reference a previously edited program, but no other program has been edited during the current editing session.

User action: Use the **New** or **GoTo** function-key command (or the N keyboard command) to change to a new program.

No picture data available **(–723)**

Explanation: A vision operation was attempted that requires processed picture data (run-length encodings) when no processed picture data was available.

User action: Issue a VPICTURE or VWINDOW command or instruction with a mode parameter of –1 or 1. This will provide the processed picture data needed for rulers or reprocessing.

No program specified **(–301)**

Explanation: No program was specified for an EXECUTE or SEE command or instruction, or DEBUG command, and no previous program is available as a default.

User action: Type the line again, providing a program name.

No prototypes **(–702)**

Explanation: This is the response to the monitor commands VSTORE and VSHOW (without a parameter) when no prototypes currently exist.

User action: Load some vision object prototypes or train a new one.

No robot connected to system **(–622)**

Explanation: An attempt has been made to attach a robot with a system that does not support control of a robot. (Note that some commands, instructions, and functions implicitly attach the robot.)

User action: Make sure the system has been booted from the correct system disk (for example, use the ID command to display the system identification). Change the program so that it does not attempt to attach the robot.

No vision system selected **(–751)**

Explanation: The current task has not selected a vision system. By default, vision system 1 is selected. This error may indicate the vision option is not installed.

User action: Use the SELECT() function to select a vision system.

***No zero index* Belt** n **(–1011)**

Explanation: The conveyor belt controller did not detect a zero-index mark for the indicated belt.

User action: Make sure the value of the BELT.ZERO.COUNT parameter is set correctly. Make sure the belt encoder is connected properly. If the problem persists, contact Adept Customer Service.

***No zero index* Mtr** n **(–1004)**

Explanation: The motor controller did not detect a zero-index mark for the indicated joint.

User action: Before you can resume running the program, you need to recalibrate the robot. If the problem persists, contact Adept Customer Service.

Nonexistent file **(–501)**

Explanation: (1) The requested file is not stored on the disk accessed. Either the name was mistyped or the wrong disk was read.

(2) The requested graphics window title, menu, or scroll bar does not exist.

User action: (1) Check the file name--use the FDIRECTORY command to display the directory of the disk.

(2) Check the name of the graphics window element specified.

***Nonexistent subdirectory* (–545)**

Explanation: The subdirectory referenced in a file specification does not exist on the disk that is referenced. Note that the subdirectory may be part of a default directory path set by the DEFAULT monitor command.

User action: Check that the subdirectory name was entered correctly. Check that the correct disk drive was referenced and that the correct diskette is loaded. Use an FDIRECTORY command to display the directory containing the subdirectory. Check that the default directory path is correct.

***Not a directory* (–567)**

Explanation: The caller specified a nondirectory in an NFS directory operation.

User action: Specify a directory in the operation, or use the correct nondirectory operation.

***Not attached to logical unit* (–516)**

Explanation: A program has attempted to perform I/O to a logical unit that it has not attached with an ATTACH instruction. Logical unit 4 allows output without being attached, but all other logical units require attachment for both input and output.

User action: Edit the program to make sure it attaches a logical unit before attempting to use it to perform I/O.

***Not configured as accessed* (–544)**

Explanation: An attempt has been made to access a serial line or other I/O device in a manner for which it is not configured. For example, a Kermit or network line cannot be accessed as a simple serial line.

User action: Check on the proper way to access the serial line for the current configuration. Use the configuration utility program to display the serial line configuration and change it if desired.

***Not enough program stack space* (–413)**

Explanation: An attempt was made to call a subroutine, process a reaction subroutine, or allocate automatic variables when the stack for the program task was too full.

User action: Reorganize the program logic to eliminate one or more nested subroutine calls or reactions; eliminate some of the automatic variables that are allocated by the programs; or use the STACK monitor command to increase the size of the stack for the program task. The program may be restarted with the RETRY command.

***Not enough prototype storage area* (–717)**

Explanation: The vision system does not have enough memory to store all of the prototypes requested.

User action: VDELETE unused prototypes, load fewer prototypes, or use simpler object models.

***Not enough storage area* (–411)**

Explanation: There is no more space in RAM for programs or variables.

User action: Delete unused programs and variables. If the memory is fragmented because of numerous deletions, it can be consolidated by issuing the commands STORE save_all, ZERO, and LOAD save_all. This writes the memory contents to the disk and reads them back into memory. Note, however, that this procedure does not retain any variables that are not referenced by any program in memory, nor does it retain the values of variables that are defined to be AUTO or LOCAL.

***Not found* (–356)**

Explanation: The search operation was unable to locate the specified string.

User action: Enter a new search string, or consider this an informational message and continue with other operations.

***Not owner* (–566)**

Explanation: The client does not have the correct access identity to perform the requested NFS operation.

User action: Modify the LOCAL_ID statement in the V+ configuration file (using the CONFIG_C utility) as required to gain access to the server. You may also need to change the access setup on the server itself.

***NVRAM battery failure* (–665)**

Explanation: The nonvolatile RAM battery backup has failed and the RAM may not hold valid data.

User action: Replace NVRAM battery.

***NVRAM data invalid* (–661)**

Explanation: The nonvolatile RAM has not been initialized or the data has been corrupted.

User action: Power down your controller and restart your system. If the error persists, contact Adept Customer Service.

***Obstacle collision detected* (–901)**

Explanation: A possible or actual collision has been detected between the robot and any statically defined obstacles. This error is similar to *Location out of range* in that it is often detected by the kinematic solution programs as the robot is moving.

User action: Move the robot away from the obstacle and continue the motion, or modify the application program to avoid the obstacle and reexecute the program.

Old value: n, New value: n

Explanation: The specified watchpoint has detected a change in value for the expression being watched. The change occurred because of execution of the program step just before the one indicated.

User action: Enter a PROCEED (Ctrl+P), RETRY, SSTEP (Ctrl+Z), or XSTEP (Ctrl+X) command to resume program execution.[1] Otherwise, enter any other monitor command.

[1] The command keys CTRL+P, CTRL+X, and CTRL+Z are accepted only while using the V+ program debugger in its monitor mode.

Option not installed **(–804)**

Explanation: An attempt has been made to use a feature of a V+ system option that is not present in this robot system.

User action: Power down the controller and try starting it again. Contact Adept Application Engineering if the problem repeats.

Out of graphics memory **(–549)**

Explanation: There is no more space in the graphics memory on the system processor for windows, icons, fonts, or other graphics items.

User action: Delete unused graphics items, or reduce the size of windows, to free up graphics memory.

Out of I/O buffer space **(–532)**

Explanation: An I/O operation cannot be performed because the V+ system has run out of memory for buffers.

User action: Delete some of the programs or data in the system memory and retry the operation. (Also see *Not enough storage area*.)

Out of vision transform memory **(–753)**

Explanation: The space allocated for vision transformations is inadequate. (A vision transformation may be defined for each task for each CPU running V+ user tasks.) Vision transformations are defined with the VTRANS instruction.

User action: Define only the vision transformations that you need. If more memory must be allocated to vision transformations, see the description of the DEVICE instruction.

Out of network resources **(–559)**

Explanation: This error applies to many circumstances. Listed below are several possible cases:

1. Too many ports are simultaneously in use for TCP and NFS; there are no more buffers available for incoming and outgoing packets.
2. Too many drives are being mounted.
3. Too many NFS calls were made simultaneously from separate tasks to a nonfunctional NFS server.

4. Too many node names are being defined.
5. An incoming IP packet was fragmented into too many pieces and V+ was unable to reassemble it. (This is a highly unlikely occurrence.)

User action: Correct the problem generating the error.

***Output record too long* (–529)**

Explanation: A TYPE, PROMPT, or WRITE instruction has attempted to output a line that is too long. The maximum line length is 512 characters.

User action: Change the program to output less information from each instruction. Remember that you can concatenate the output from separate instructions by using /S to suppress the carriage return and line feed normally done at the end of each TYPE output.

***Overtravel* Mtr** n **(–1034)**

Explanation: The indicated motor has moved beyond the hardware-limited range of motion.

User action: Move the robot back into the working envelope. Correct whatever caused the robot to get into the restricted area. Then enable power.

***PANIC command* (–633)**

Explanation: The operator has entered a V+ PANIC monitor command which has stopped the current robot motion. High power is still enabled.

User Action: To continue the current motion, enter the RETRY monitor command. To continue after the current motion, enter the PROCEED monitor command.

(PAUSED) (9)

Explanation: A PAUSE instruction has been executed, and thus the current program has suspended execution.

User action: Any monitor command can be entered. To continue execution of the program, type **proceed** followed by the task number if it is not 0.

***Position out of range* Jt** n **(–1002)**

Explanation: (1) The requested motion was beyond the software-limited range of motion for the indicated joint; (2) while enabling high power, V+

detected that the indicated robot joint was outside the software limit.

User action: (1) Modify the program as required to prevent the invalid motion request. (Because the robot did not actually move out of range, you do not need to move the robot before continuing); (2) move the robot back into the working envelope. Correct whatever caused the robot to get into the restricted area. Then enable power.

***Position out of range* Mtr** n **(–1023)**

Explanation: (1) The requested motion was beyond the software-limited range of motion for the indicated motor; (2) while enabling high power, V+ detected that the indicated robot motor was outside the software limit.

User action: (1) Modify the program as required to prevent the invalid motion request (Because the robot did not actually move out of range, you do not need to move the robot before continuing.); (2) move the robot back into the working envelope. Correct whatever caused the robot to get into the restricted area. Then enable power.

***Positive overtravel* Mtr** n **(–1033)**

Explanation: The indicated motor has moved beyond the hardware-limited positive range of motion.

User action: Move the robot back into the working envelope. Correct whatever caused the robot to get into the restricted area. Then enable power.

***Power disabled: Manual/Auto changed* (–645)**

Explanation: V+ disables power when the VFP switch moves from MANUAL to AUTO or vice versa.

User action: Use any valid method to enable high power.

***Power failure detected* (–667)**

Explanation: Indicates that a controller AC power-fail condition has been detected. If battery backup is installed, this error will be reported (when power is restored) by any I/O operations that were canceled due to the power failure. This error code may be trapped by a program using the REACTE instruction in order to provide some level of automatic power failure response.

User action: The user may need to restart or repeat any operations that were interrupted by the controller AC power failure. Some reinitialization of the system may be required: for example, any robot(s) connected to the controller need to be recalibrated after a controller power failure.

***Power failure detected by robot* (–632)**

Explanation: Indicates that a controller power failure condition has been detected by the robot control software while a robot is attached to a program. This error is issued in addition to –667 if a program has a robot attached and has a REACTE routine defined. Unlike error –667, if no REACTE routine is defined and a robot is attached, the V+ program stops with this error.

User action: The user may need to restart or repeat any operations that were interrupted by the controller AC power failure. Some reinitialization of the system may be required: for example, any robot(s) connected to the controller will need to be recalibrated after a controller power failure

Press HIGH POWER button to enable power (57)

Explanation: The HIGH POWER ON/OFF button on the front panel must be pressed to complete the process of enabling high power.

User action: When the HIGH POWER ON/OFF button on the VFP blinks, promptly press the button to complete the two-step process of enabling high power. (You must press the button within the time period specified in the V+ configuration data.)

***Processor crash* CPU = n (None)**

Explanation: V+ has detected that the specified CPU within the controller has entered a fatal error state. A crash message from that processor is displayed immediately following. A software error or hardware problem with that processor is likely.

User action: It would be appreciated if you would report the error to Adept Application Engineering. Please include the details of the error message and exactly what you were doing at the time the error occurred. You should store the programs that are in memory, power down the controller, and start it again. (If the processor ID shown is 1, you can restart V+ by pressing CTRL+G. The robot servos will not function, but you can STORE the programs in memory.) If the problem persists, contact Adept Customer Service.

***Program already exists* (–309)**

Explanation: An attempt has been made to LOAD a program that already exists, or to COPY or RENAME a program to a name that is already in use.

User action: Delete the conflicting program or use a different name.

***Program argument mismatch* (–408)**

Explanation: The arguments in a CALL, CALLS, or EXECUTE instruction do not match the arguments in the program being referenced because they are of different types.

User action: Modify the CALL, CALLS, or EXECUTE instruction, or the .PROGRAM statement of the referenced program, so that the argument types match. If arguments are omitted in the CALL, CALLS, or EXECUTE instruction, make sure the appropriate commas are included to position the arguments that are present.

Program completed (3)

Explanation: The program has been executed the number of times specified in the last EXECUTE command or instruction.

User action: Any monitor command can be entered, except that PROCEED cannot be used to resume program execution.

Program program_name doesn't exist. Create it (Y/N)? (None)

Explanation: An attempt has been made to use the SEE editor to access a program that does not currently exist.

User action: Enter a Y to have the program created. Any other input, including just pressing RETURN, cancels the edit request.

Program HOLD (15)

Explanation: The RUN/HOLD button on the pendant has been pressed while a robot program was executing, and it is now suspended.

User action: Any monitor command can be entered. To continue execution of the program, type **proceed** or **retry**, or press the PROGRAM START button on the controller. (The RUN/HOLD button can be held down to temporarily resume execution of the program if the front-panel keyswitch is in the MANUAL position.)

***Program interlocked* (–308)**

Explanation: An attempt has been made to access a program that is already in use by some V+ process. For example, you have attempted to delete or edit a program that is being executed, or execute a program that is being edited.

User action: Abort the program or exit the editor as appropriate and retry the operation. You can use the SEE editor in read-only mode to look at programs that are interlocked from read-write access.

Program name? (None)

Explanation: A SEE editor command to change to a different program has been entered.

User action: Enter the name of the new program to be edited, or press RETURN to cancel the request.

***Program not executable* (–307)**

Explanation: Because of program errors detected during loading or upon exiting from the editor, this program cannot be executed.

User action: Edit the program to remove any errors.

***Program not on top of stack* (–421)**

Explanation: A DO context specification has referenced an automatic variable or a subroutine argument in a program that is not on the top of the stack for the specified task.

User action: Reenter the DO command and specify the correct program context or eliminate references to automatic variables and subroutine arguments. Use the STATUS command to determine which program is on the top of the stack.

Program task # stopped at program_name, step step_number date time (4)

Explanation: Execution of the program task indicated by # has terminated for the reason indicated in the message that preceded this message. The step number displayed corresponds to the next NEXT program step that would be executed (for example, if PROCEED were entered). The current date and time are displayed if the system date and time have been set.

User action: None. This is only an informational message.

Program task not active **(–318)**

Explanation: An attempt was made to abort a task that was not active.

User action: None required if the correct task number was specified. Otherwise, use the STATUS command to determine which task number should have been used.

Program task not in use **(–319)**

Explanation: A program task cannot be accessed because it has never been used. (Such program tasks do not use any system memory and do not appear in the STATUS display.)

User action: None.

Protected program **(53)**

Explanation: An attempt has been made to list a program that is protected from user access.

User action: None.

Protection error **(–530)**

Explanation: An I/O operation cannot be performed because (1) it attempted to write to a disk that is write protected, or (2) the user does not have the proper access status.

User action: Check the diskette to make sure the write-protect tab is in the correct position. Use an FDIRECTORY command to display the disk directory. If the file has protected (P) or read-only (R) protection, you cannot access it in the way attempted.

Recursive macros illegal **(–357)**

Explanation: An attempt was made to execute a macro recursively. That is, the macro contained a command character sequence that (directly or indirectly) restarted execution of the macro.

User action: Change the macro definitions as necessary to make sure neither macro invokes itself. You can have the U macro invoke the Y macro, or vice versa (but not both).

Region too big **(–743)**

Explanation: While using optical character recognition (OCR) to recognize text (VOCR) or train a font (VTRAIN.OCR), a region in the given window was more than 63 pixels in the horizontal or vertical dimension.

User action: Make sure there are no extraneous regions in the training window. If the characters in the font are too large, use a camera lens with a shorter focal length, or increase the distance between the camera and the text.

Region too complicated **(–744)**

Explanation: While using optical character recognition (OCR) to train a font (VTRAIN.OCR), a character region was encountered with more than 20 concavities and holes.

User action: Look at the binary image (with VDISPLAY mode 2). Perhaps the threshold needs adjustment.

Remote has not exported network resource **(–563)**

Explanation: The NFS server has not exported the designated path for use by clients.

User action: Check the NFS server setup, and check the path that the V$^+$ system uses.

Reserved word illegal **(–457)**

Explanation: An attempt has been made to use a V$^+$ reserved word for a variable name. (See Tables **1-1** to **1-9** for a list of the reserved keywords.)

User action: Use a different name for the variable. You can, for example, append a prefix or suffix to the attempted name.

Return manual control pendant to background display (^C to exit) **(None)**

Explanation: The manual control pendant display must be in background mode for the operation you have selected.

User action: Press the DONE button on the pendant one or more times to exit the current function.

Robot already attached to program **(–602)**

Explanation: A program has executed more than one ATTACH instruction for the robot, without executing a DETACH in between. Or an attempt has been made to SELECT another robot when one is already attached. The robot is still attached even after this error occurs.

User action: Check the program logic—remove redundant ATTACH instructions, or DETACH the current robot before attempting to SELECT another robot.

Robot interlocked **(–621)**

Explanation: (1) An attempt has been made to access a robot or external device that is already being used by a different program task or by the system monitor; (2) an attempt has been made to calibrate the robot with the VFP keyswitch set to MANUAL.

User action: (1) Review the program logic and make sure the robot or device is being controlled by only one program task; (2) with V^+ 11.3 (and later), you cannot calibrate the robot when the VFP keyswitch is set to MANUAL. (This is for safety reasons, and also to avoid triggering the accelerometer when calibrating.) Set the keyswitch to AUTO before attempting calibration.

Robot module not enabled **(–900)**

Explanation: The indicated robot module is present in memory, but it was not enabled for use due to an error (which is reported by a separate message).

User action: Use the CONFIG_C and/or SPEC utilities to correct the module configuration.

***Robot module not loaded* ID:** n **(–628)**

Explanation: This error occurs only during startup when a robot module has been configured using the CONFIG_C utility, but the robot module is not present in memory.

User Action: Use the CONFIG_C utility to add the robot module to the boot disk before rebooting.

Robot not attached to this program **(–601)**

Explanation: An attempt has been made to execute a robot-control command or instruction in one of the following invalid situations:

(1) The system is not configured to control a robot. (2) There is no robot connected to the system. (3) The robot is attached to a different program task.

User action: (1) Make sure the system is booted from the proper system disk, or remove the robot-control instruction.

(2) Connect the robot or enable the DRY.RUN system switch.

(3) Modify the program logic as required to ensure that only one program task is controlling the robot at any given time.

***Robot not calibrated* (–605)**

Explanation: An attempt has been made to execute a robot-control program when the robot is not calibrated. No motion is allowed until the robot is calibrated.

User action: If you want to use the robot, issue a CALIBRATE command or have your program execute a CALIBRATE instruction. Or enable the DRY.RUN switch to allow program execution without using the robot.

***Robot power off* (–604)**

Explanation: High power is not turned on or cannot be turned on because of a hardware failure. On a system with the Manual Mode Safety Package (MMSP), you can get this error if you press the HIGH POWER ON/OFF button before it starts to flash.

User action: (1) Turn on high power and reenter the last command; (2) on a system with the MMSP, wait for the HIGH POWER ON/OFF button to start to flash before you press it.

***Robot power on* (–627)**

Explanation: An attempt has been made to perform an action that requires high power to be off.

User action: DISABLE POWER and reexecute the action.

***RSC bad packet format* (–655)**

Explanation: V^+ has received an incorrect data packet from the robot signature card, during the initial calibration data load.

User action: None unless the calibration load fails. If the problem persists, contact Adept Customer Service.

***RSC calibration load failure* (–656)**

Explanation: V+ cannot load calibration data from the robot signature card (RSC).

User action: Power down the controller and make sure the robot cables are correctly and securely connected. If the problem persists, contact Adept Customer Service.

***RSC communications failure* (–651)**

Explanation: V+ has lost communications with the robot signature card (RSC). Either a hardware problem has occurred or the robot is being operated in an environment with excessive electrical noise.

User action: Check the connections of the robot cables. Turn high power back on, calibrate the robot, and resume program execution. If the problem persists, contact Adept Customer Service.

***RSC hardware failure* (–669)**

Explanation: The RSC has reported an internal failure. Because RSC failures almost always cause the RSC to stop communicating altogether (rendering it incapable of reporting the failure), this error message may be due to some other cause, such as electrical noise at the RSC or within or around the arm signal cable.

User Action: If the problem persists, contact Adept Customer Service.

***RSC module ID doesn't match robot* (–676)**

Explanation: The V+ configuration data contains an explicit ID specification for a robot module (for example, 6 for the Adept 550 robot), and the robot RSC does not contain that ID number.

User Action: Make sure that the correct type of robot is being used. Use the CONFIG_C utility to change the module ID to –1 in the V+ configuration data. Correct the module ID in the RSC.

***RSC power failure* (–670)**

Explanation: The RSC has reported that its power is failing. Because a power failure on the RSC almost always causes it to stop communicating altogether (rendering it incapable of reporting the failure), this error message may be due to some other cause, such as electrical noise at the RSC or within or around the arm signal cable.

It is possible that the power lines to the RSC have an intermittent connection somewhere.

User Action: If the problem persists, contact Adept Customer Service.

***RSC reset* (–652)**

Explanation: V+ has detected that the robot signature card (RSC) has lost power temporarily, but is now functioning.

User action: Turn high power back on and resume program execution. If the problem persists, check the cabling to the robot. Contact Adept Customer Service if no solution can be found.

***RSC time-out* (–653)**

Explanation: V+ has not received a response from the robot signature card (RSC) when expected, during the initial calibration data load. The RSC or its cabling is probably faulty.

User action: Power down the controller and check the cables to the robot. If the problem persists, contact Adept Customer Service.

***RSC transmission garbled* (–654)**

Explanation: V+ has received an invalid transmission from the robot signature card (RSC). Either a hardware problem has occurred or the robot is being operated in an environment with excessive electrical noise.

User action: None unless the calibration load fails or RSC communications fail. If the problem persists, contact Adept Customer Service.

Searching for string (exact case) (None)

Explanation: The SEE editor command 0' has been entered. The editor is prepared to search for the string indicated, in the search mode indicated.

User action: This is an informational message. You can use the **Repeat** command to perform the indicated search, or you can use **Find** (or **Change**) to initiate a new search (or replacement) operation. The EXACT extended command controls the setting of the search mode.

Searching for string (ignoring case) (None)

Explanation: The SEE editor command 0' has been entered. The editor is prepared to search for the string indicated, in the search mode indicated.

User action: This is an informational message. You can use the **Repeat** command to perform the indicated search; or you can use **Find** (or **Change**) to initiate a new search (or replacement) operation. The EXACT extended command controls the setting of the search mode.

***Servo board E-Stop fuse open* (–673)**

Explanation: Your servo board has a fused ESTOP circuit, and the system has detected an open circuit at that location.

User Action: Refer to your hardware documentation, consult with Adept Customer Service as needed for details about types and locations of fuses, and replace the fuse.

***Servo board 12v fuse open* (–671)**

Explanation: Your servo board has a fused 12-volt bus, and the system has detected an open circuit at that location.

User Action: Refer to your hardware documentation, and replace the fuse.

***Servo board solenoid fuse open* (–672)**

Explanation: Your servo board has a fused robot solenoid control line, and the system has detected an open circuit at that location.

User Action: Refer to your hardware documentation, and replace the fuse.

***Servo task overloaded* (–674)**

Explanation: A servo interrupt task has used up all the execution time. The detection algorithm reports an error when the servo interrupt task completely occupies 10 or more time slices per second of real time. The robot went to a fatal error state when this error occurred, and the servo interrupt task stopped running.

User action: Change one or more of the following: (1) move servo tasks off CPU #1 to allow more time for trajectory generation, (2) change CPU #1 from an 030 to an 040 to increase the throughput, or (3) reduce the number of robots or axes that you are operating.

Set for CASE DEPENDENT searches (None)

Explanation: The EXACT extended command has been used to change the method by which character case is considered during string searches. The message indicates how case will be considered in subsequent searches (for the current or future search-for strings).

User action: None. This is an informational message.

Set for CASE INDEPENDENT searches **(None)**

Explanation: The EXACT extended command has been used to change the method by which character case is considered during string searches. The message indicates how case will be considered in subsequent searches (for the current or future search-for strings).

User action: None. This is an informational message.

***Skew envelope error* Mtr** n **(–1022)**

Explanation: The two motors associated with a split robot axis were not tracking each other with sufficient accuracy.

User action: Make sure nothing is obstructing the robot motion. Turn on high power and try to perform the motion at a slower speed. If necessary, use the SPEC utility to increase the maximum skew error.

Soft envelope error **(–1006)**

Explanation: The indicated motor was not tracking the commanded position with sufficient accuracy, indicating a failure in the hardware servo system or something impeding the path of the robot. Because this was not considered a serious error, a controlled motion stop occurred and high power remains on.

User Action: Try to perform the motion at a slower speed. Make sure that nothing is obstructing the robot's motion. If the error recurs, contact Adept Customer Service.

Software checksum error **(–316)**

Explanation: During processing of a FREE command the V^+ system has detected a checksum error in the system memory. This indicates a problem with the system software or hardware. (Note, however, that a checksum error will be introduced if any patches are made to the system software after the system is loaded from disk and started up.) The following codes are appended to the message indicating where the error occurred: Os, operating system; V^+, V^+ interpreter or trajectory generator; Vi, vision software; Sv, servo software.

User action: Report to Adept Application Engineering the error and information about any possible contributing circumstances. You can continue to use the system, but you should keep in mind the possibility of a problem with the hardware.

***Software incompatible* Code** n **(–1026)**

Explanation: The servo code has detected an incompatibility between the servo code and calibration software.

User action: Make sure that you are using the calibration software (in the \CALIB\ directory) that you received with the V+ system you are using. If you are using the correct software, note the code number, and call Adept Customer Service.

Speed pot or STEP not pressed **(–620)**

Explanation: While the VFP was set to MANUAL mode, a V+ program tried to initiate a robot motion, but you failed to press the STEP button and speed bar on the MCP.

User action: When the VFP is set to MANUAL mode and a V+ program is about to initiate robot motions, press the STEP button and speed bar on the MCP. To continue the motion once it has started, you can release the STEP button but must continue to press the speed bar.

SPIN motion not permitted **(–638)**

Explanation: Either a SPIN instruction has attempted to move a joint that has not been configured with the continuous-rotation capability or the robot is currently tracking a belt or moving under control of an ALTER instruction.

User action: Configure the joint with continuous-rotation capability, or complete the belt tracking or ALTER instruction before attempting to execute the SPIN instruction.

Step syntax MUST be valid **(None)**

Explanation: The SEE editor's AUTO.BAD extended command has been used to change the action to be taken when an invalid line is detected while editing. Subsequently, the editor will require that such a line be corrected before you will be able to perform any operation that would move the cursor off the bad line.

User action: None. This is an informational message.

Stop-on-force triggered **(–623)**

Explanation: A force-sensor Guarded Mode trip occurred when the robot was not under program control.

User action: High power must be reenabled before robot motion may continue. If the trip was not desired, make sure that Guarded Mode is disabled before the program relinquishes control of the robot to the manual control pendant.

***Stopped due to servoing error* (–600)**

Explanation: Program execution has stopped because of one or more servo errors.

User action: Correct the source of the reported servo errors, referring to your system hardware manual as required.

***Storage area format error* (–305)**

Explanation: During execution of a FREE command, V^+ has detected that programs or data in RAM may have been corrupted. This may have been caused by a momentary hardware failure or a software error.

User action: Attempt to save as much as possible onto the disk. Then enter a ZERO command or power down the controller and restart the system.

***Straight-line motion can't alter configuration* (–612)**

Explanation: A change in configuration was requested during a straight-line motion. This is not allowed.

User action: Delete the configuration change request, or use a joint-interpolated motion instruction.

***String too short* (–417)**

Explanation: A program instruction or command expected a string argument with a certain minimum length and received one that was too short.

User action: Review the syntax for the program instruction and edit the program to pass a string of the correct length.

***String variable overflow* (–416)**

Explanation: An attempt has been made to create a string value that is greater than the maximum string length of 128 characters.

User action: Edit the program to generate strings of the proper length.

Subdirectory in use **(–547)**

Explanation: An attempt has been made to delete a subdirectory that still contains files or that is being referenced by another operation (for example, an FDIRECTORY command).

User action: Check that all the files within the subdirectory have been deleted. Check that no other program tasks are referencing the subdirectory. Retry the delete operation.

Subdirectory list too long **(–546)**

Explanation: A directory path contains too many subdirectories, or the directory path is too long to be processed. The path is a combination of subdirectories in the file specification and the default directory path set by the DEFAULT monitor command. Directory paths are limited to a total of 16 subdirectories and 80 characters (including any portion of the directory path specified by the current default path).

User action: Specify a shorter directory path in the file specification or in the DEFAULT command. If you are accessing a foreign disk that contains more than 16 nested subdirectories, you cannot read the files in subdirectories nested deeper than 16 levels. In that case you will need to use the system that created the disk to copy the files to a directory that is nested less deeply.

Switch can't be enabled **(–314)**

Explanation: An ENABLE command for a certain switch has been rejected because of some error condition. For example, ENABLE POWER will fail if the system is in FATAL ERROR state.

User action: Review the description for the switch you are trying to enable, correct the error condition, and try again.

SYSFAIL asserted **(–629)**

Explanation: A board on the VME bus has encountered a severe error and asserted SYSFAIL which turns off high power. If that error is seen, it indicates a transient SYSFAIL signal or a problem with either the motion interface board or the SIO module.

User action: Restart the system. Check for proper seating of the system boards and correct device connections to the boards. Test the system with as many boards removed as possible, adding boards back in until the problem board is identified. If the problem persists, contact Adept customer service.

Task = **(None)**

Explanation: The SEE editor DEBUG extended command has been used to initiate a program debugging session for the current program. The debugger needs to know which program task you want to use when executing the program.

User action: Enter the desired task number, or press RETURN to access the same task used for the last debugging session.

Template already defined **(–748)**

Explanation: When defining a new correlation template with the program instruction VTRAIN.MODEL, the number of an existing template was given.

User action: Delete the existing template if it is no longer needed, or use a different number in the VTRAIN.MODEL instruction.

Template of uniform intensity **(–746)**

Explanation: When defining a correlation template with the VTRAIN.MODEL program instruction, the area of the image within the given template bounds has uniform intensity. Image templates must have some variation in brightness. (That is, there must be some features in the template to correlate with later.)

User action: Check the position of the template in the image and make sure it is in the desired place. Also, view the grayscale image in the current frame to make sure it is valid. (For example, maybe a strobe light did not fire, or the lens cap is still on the camera.)

Template not defined **(–747)**

Explanation: The correlation template referenced in a VCORRELATE, VDELETE, VSHOW.MODEL, or VSTORE operation does not exist.

User action: Check the correlation number supplied to the operation to make sure it is correct. Use the Models pull-down menu in the vision window (or the VSHOW.MODEL program instruction) to get a list of the templates currently defined in the vision system.

***Time-out nulling errors* Mtr** n **(–1003)**

Explanation: The indicated motor took too long to complete the last motion, possibly because the robot is blocked and cannot reach its destination.

User action: Turn on high power and retry the motion after making any necessary program changes. If this error occurs repeatedly, contact Adept Application Engineering for assistance.

***Time-out enabling amplifier* Mtr** n **(–1009)**

Explanation: The power amplifier for the indicated motor has signaled a fault condition. A momentary power failure or a hardware error may have occurred.

User action: Turn high power back on and restart the program. If the error persists, contact Adept Customer Service.

Timeout enabling power **(–675)**

Explanation: High power did not enable within the allowed amount of time, and the servos reported no other error during the timeout period.

User action: For non-Adept robots, use the SPEC utility to increase the value of the high power time-out.

For Adept robots, double-check your installation (cabling, AC power line voltages, circuit breakers, amplifier retaining screws, cables, and contactors). For information about the correct configuration for installation, refer to your Robot Instruction Handbook. Make sure that the amplifier chassis is properly connected to a power source and is turned on. Try again. If the problem persists, contact Adept Customer Service.

Timeout: Hold-to-run not toggled **(–649)**

Explanation: V+ did not enable high power because you failed to toggle properly the HOLD-TO-RUN switch on the manual control pendant.

User action: Do one or more of the following: (1) when toggling the HOLD-TO-RUN switch, release it for a minimum of about two seconds and a maximum of ten seconds, and then press it back in; and (2) make sure that you are pressing the HOLD-TO-RUN switch and not the RUN/HOLD button by mistake.

Too many arguments **(–553)**

Explanation: Too many arguments were specified for the last command or instruction.

User action: Reenter the command or instruction but with the correct number of arguments.

Too many array indices **(–474)**

Explanation: The specification of an array element contains more than three indexes.

User action: Reenter the line with the correct number of indexes.

Too many closeable windows **(–554)**

Explanation: The names of too many graphics windows have been specified to appear in the pull-down under the Adept logo in the status line at the top of the screen.

User action: Specify all subsequent windows as /NOCLOSEABLE, or delete some existing windows that appear in this pull-down.

Too many icons **(–556)**

Explanation: An attempt has been made to define more graphic icons than the system is configured to support.

User action: Delete any icons that are no longed needed. If necessary, use the CONFIG_C utility program to reconfigure your V+ system to support more icons.

Too many network errors **(–536)**

Explanation: (1) The number of errors detected by the DDCMP protocol has exceeded the maximum allowed. The local protocol is stopped, and all pending I/O requests are completed with this error.

(2) The V+ Kermit driver experienced more errors than permitted by the KERMIT.RETRY parameter.

User action: (1) Use the NET monitor command to determine the type of errors that have occurred. Check for noise on the communication line, errors in the remote DDCMP implementation, or program logic that sends messages faster than they can be processed. Use the appropriate FCMND instruction to increase the maximum number of errors.

(2) Set the KERMIT.RETRY parameter to a larger value, increase the retry threshold on the remote server, restart the Kermit session, and retry the operation that failed.

***Too many vision requests pending* (–703)**

Explanation: A program has issued too many VLOCATE commands before the first ones have completed.

User action: Edit the program to wait for pending VLOCATE requests to complete before issuing more.

***Too many windows* (–550)**

Explanation: An attempt was made to create a graphics window when the maximum number of windows were already defined. (The V+ system uses two windows for the screen and the top status line. Every title bar, menu bar, and scroll bar is a separate window. The pull-down window is always allocated even if it is not visible. Systems with AdeptVision always have the vision-training window allocated.)

User action: Where possible, change your window definitions to omit menu bars and scroll bars. If necessary, use the utility program CONFIG_C to increase the number of window buffers.

***Trajectory clock overrun* (–636)**

Explanation: One of these three conditions has occurred: (1) the time for a new trajectory point has arrived, but the internal trajectory task has not finished computing the previous point; (2) the servos did not receive trajectory data at the expected time because the trajectory task took too long to compute and write out the data; or (3) the trajectory interval is equal to or less than the servo interval.

User action: Perform one or more of the following: (1) if the trajectory cycle time is less than 16msec change it to the next longer time; (2) move servo tasks off CPU #1 to allow more time for trajectory generation; (3) change CPU #1 from an 030 to an 040 to increase the throughput; (4) reduce the number of robots or axes that you are operating; or (5) if the trajectory cycle time is set to 2ms, make sure the servo interval is 1ms.

***Undefined program or variable name* (–406)**

Explanation: The program or variable, referenced in a command or program step, does not exist—possibly because the name was mistyped.

User action: If the correct name was entered, create the program or variable using one of the V+ editors or the appropriate V+ monitor commands, or by loading from a disk file.

***Undefined value* (–401)**

Explanation: (1) A variable has been referenced that has not been assigned a value.

(2) Using the SEE editor, an attempt has been made to use a macro, return to a memorized cursor position, or perform a repeat string search or change without first performing the appropriate initialization sequence.

User action: (1) Assign the variable a value or correct its name.

(2) Define the macro, record a cursor position, or enter the desired search/replacement string(s).

***Undefined value in this context* (–420)**

Explanation: An automatic variable or subroutine argument value appears in a monitor command, but the specified program is not on the execution stack for the specified program task. Automatic variables and subroutine arguments have values only when the program that defines them is on a stack.

User action: Change the monitor command to not reference the variables. Check that the program is on the expected execution stack. You can place a PAUSE instruction or breakpoint in the program to stop it while it is on the execution stack.

***Unexpected end of file* (–504)**

Explanation: (1) If a file was being loaded from the disk, the end of the file was encountered unexpectedly.

(2) If a program is reading a file, this error code merely indicates that the end of the file has been reached and should not be interpreted as a real error.

(3) This message results if a CTRL+Z is pressed in response to a program PROMPT.

(4) A break condition was detected on a serial line.

User action: (1) Try again to read the file.

(2) Close the file and continue program execution.

(3) Treat the program as having been aborted early by user request.

Unexpected text at end of line **(–451)**

Explanation: The previous command or instruction could not be recognized by V+, possibly because of a mistyped function name or because an argument was specified where none is allowed.

User action: Reenter the line, correcting the syntax error.

***Unexpected zero index* Belt** n **(–1012)**

Explanation: A zero index signal was received from the encoder for this motor belt at an unexpected time. The encoder may be gaining or losing counts, there may be a hardware problem with the zero index signal, or the Counts per zero index configuration parameter may be set incorrectly.

User action: Continue to use the system. Contact Adept Customer Service if this error occurs repeatedly.

***Unexpected zero index* Mtr** n **(–1005)**

Explanation: A zero index signal was received from the encoder for this motor at an unexpected time. The encoder may be gaining or losing counts, there may be a hardware problem with the zero index signal, or the Counts per zero index configuration parameter may be set incorrectly.

User action: Turn on high power, calibrate the robot, and continue to use the system. If this error occurs repeatedly, contact Adept Customer Service.

Unknown editor command **(–363)**

Explanation: An unknown keystroke or extended command was issued while using the SEE program editor.

User action: Enter another command.

Unknown error code **(–800)**

Explanation: An error code that does not correspond to a known error message was received by V+ from an external device.

User action: If an external computer is communicating with V+ when the error occurs, verify that it is sending proper error codes. Otherwise, a software error is indicated. It would be appreciated if you would report the error to Adept Application Engineering. Please include

the details of the error message and exactly what you were doing at the time the error occurred.

***Unknown function* (–462)**

Explanation: While accepting a program statement, V+ has encountered a reference to a function that it does not recognize. This could be due to a mistyped function name or the leaving out of an operator between a symbol and a left parenthesis.

User action: Check the spelling and syntax and reenter the line.

***Unknown instruction* (–452)**

Explanation: An instruction was entered (or read from a disk file) that was not recognized by the system. This error is often caused by mistyping the instruction name, or trying to use a command as an instruction or vice versa. Note that statements with errors are turned into bad lines beginning with a question mark.

If the message occurred while loading a file from the disk, the file was probably created off-line, or with a different V+ system (different version or options), and the indicated line is not compatible with the V+ system in use.

User action: Correct the line or enter it again, making sure the spelling and usage are correct. When using the SEE editor, an invalid statement is either converted to a bad line or must be corrected before you can leave that line (depending on the setting of the AUTO.BAD feature). In the case of an error while loading from the disk, edit the program to correct the indicated instruction.

***Unknown keyword* (–424)**

Explanation: The keyword in an FSET instruction is unknown in the context in which it was found. (Most often, a keyword used for a serial line was used when referencing a window or vice versa.)

User action: Correct the line in the executing program or reenter the command with the correct keyword.

***Unknown model* (–759)**

Explanation: The VPLAN.FINDER vision instruction was given the name of a model that does not exist on the system.

User action: Supply the name of an existing model to the VPLAN.FINDER vision instruction.

***Unknown network node* (–537)**

Explanation: A reference has been made to a network node address that is not known by the local network.

User action: Check that the correct node address was specified. Check that the remote node is active and connected to the network. If explicit routing tables are used, check that they specify this node.

***Unknown prototype* (–707)**

Explanation: A vision command or instruction has referenced an object prototype that is not known to the vision system. This may be due to mistyping the prototype name.

User action: Enter the command VSHOW at the terminal for a list of the known prototypes. If necessary, load the appropriate prototype file from disk or VTRAIN the prototype.

***Unknown sub-prototype* (–731)**

Explanation: A vision command or instruction has referenced an object subprototype that is not known to the vision system. This may be due to mistyping the prototype name.

User action: Use the command VSHOW at the terminal to display the subprototypes defined for the specified prototype. If necessary, load the appropriate prototype file from disk or use VDEF.SUBPROTO to define the sub-prototype.

***User has not tested Cat3 system* (–648)**

Explanation: A system with the EN954 Safety Category 3 option—the Manual Mode Safety Package (MMSP)—has not been successfully commissioned with the SAFE_UTL utility program.

User action: Test the MMSP with the SAFE_UTL utility before enabling power for the first time. Adept recommends that you rerun the utility program every three months. If you have connected the robot to a different controller or replaced the controller or the SIO module, repeat the test. (For information on the use of SAFE_UTL, refer to the AdeptOne-MV/AdeptThree-MV Robot Instruction Handbook.)

Variable type mismatch **(–465)**

Explanation: One or more of the variables in the line is of a type inconsistent with the other variables or with the type required by the command or instruction. For example, you may be trying to mix location variables with real-valued variables. If this error occurs upon exiting from the editor, the variable type within the program conflicts with the type of a global variable that is already defined.

User action: Check the syntax for the operation and reenter the line, correcting the mismatch. Delete conflicting global variables, if appropriate.

Vision aborted **(–749)**

Explanation: (1) The Abort menu item in the vision window has been selected. If a vision instruction in a V+ program was being executed, it is aborted and the error code for this message is returned (for access with the ERROR function); (2) A V+ program has been aborted when it was executing a vision instruction. (In this case, the error code for the standard Aborted message is normally returned.); (3) a VABORT instruction was issued.

User action: If you selected the Abort menu item by mistake, you can make the V+ program continue by typing **retry** *n* on the keyboard, where n is the number of the task that stopped. Typing **proceed** *n* also resumes program execution, but the aborted vision instruction is not retried.

***[Vision error] <details>** **(None)**

The following vision error messages can be displayed on the V+ system terminal any time while the VISION system switch is enabled. When one of these errors occurs, all pending vision commands (for example, VLOCATE) are aborted and the VISION switch is disabled. The user must reenable the VISION switch in order to resume using the vision system. Prototypes are not deleted from memory when these errors occur, or when the VISION switch is enabled.

Each of these messages has the form *[Vision error] <details>*, where <details> represents specific information identifying the error. That information will help Adept personnel to identify the specific nature and cause of the error.

[Vision error] Bit-masking failed on FS #n **(None)**

[Vision error] Bit-packer failure, n bus errors instead of 2 **(None)**

[Vision error] Bit-packer returned wrong data **(None)**

[Vision error] Bus error reading video FBn a ... at a time **(None)**

[Vision error] Bus error writing video FBn a ... at a time **(None)**

[Vision error] Bus error reading video ILUTm #n **(None)**

[Vision error] Bus error writing video ILUTm #n **(None)**

[Vision error] Bus error reading video register aaa **(None)**

[Vision error] Bus error writing video register aaa **(None)**

Explanation: A failure of the vision hardware has occurred.

User action: See the general comments above. If the problem persists, contact Adept Customer Service and provide the exact details of the error message.

[Vision error] Camera, multiplexor, or frame grabber failure #n **(None)**

Explanation: Generally this error indicates that a failure of the vision hardware has occurred.

User action: See the general comments above. If the problem persists, contact Adept Customer Service and provide the exact details of the error message.

[Vision error] Internal confusion #n **(None)**

Explanation: This error message should never appear.

User action: See the general comments above. If the problem persists, contact Adept Application Engineering and provide the exact details of the error message.

[Vision error] No acquire interrupt at level 4 **(None)**

Explanation: A failure of the vision hardware has occurred.

User action: See the general comments above. Contact Adept Customer Service if the problem persists.

[Vision error] Out of memory #n **(None)**

Explanation: Either a fixed allocation of memory has been depleted (for example, run lengths) or the general memory allocator has run out of RAM. A common cause of this error is the processing of an overly complex binary image. In this case the display should show many small regions (noise), often due to a bad threshold value.

User action: See the general comments above. If the problem persists, contact Adept Application Engineering and provide the exact details of the error message.

[Vision error] Read (...) of video FBn different than data written **(None)**

Explanation: A failure of the vision hardware has occurred.

User action: See the general comments above. If the problem persists, contact Adept Customer Service and provide the exact details of the error message.

[Vision error] Video ILUTm #n read different than write **(None)**

Explanation: A failure of the vision hardware has occurred.

User action: See the general comments above. If the problem persists, contact Adept Customer Service and provide the exact details of the error message.

Vision not calibrated **(–713)**

Explanation: A vision command was entered that required the vision system to be calibrated, and the vision system is not calibrated.

User action: Calibrate the vision system or load calibration data from a disk file.

VISION not enabled **(–701)**

Explanation: A vision command was entered before the vision system has been enabled.

User action: Enter an ENABLE VISION command and retry the previous command.

***Vision option not installed* (–720)**

Explanation: During initialization, the V+ system failed to detect the presence of the vision processor. No vision instructions or commands will be accepted. Otherwise, V+ will operate normally.

User action: Check to make sure that the vision processor is installed and that your software supports vision. Power down the controller and restart. If the problem persists, contact Adept Customer Service.

***Vision system out of memory* (–733)**

Explanation: The vision system has run out of free memory for the last operation attempted. This message should not be confused with *[Vision error] Out of memory*. This error does not disable the vision switch and is always returned in direct response to the last vision instruction.

User action: Reduce the complexity of the image or reduce the number of models in memory. If the problem persists, contact Adept Customer Service.

***Warning* Monitoring watchpoint (55)**

Explanation: Program execution has begun while a watchpoint is set.

User action: None. This is an informational message. You may want to disable the watchpoint to eliminate its slowing down of program execution.

***Warning* Not calibrated (51)**

Explanation: The robot servo system and joint position sensors are not calibrated. Thus, any location variables that are defined may not represent the locations desired.

User action: Enter a CALIBRATE command or have your program execute a CALIBRATE instruction.

***Warning* Protected and read-only programs are not stored (52)**

Explanation: A STORE command has been executed while protected and/or read-only programs are loaded in the V+ system memory. The protected and read-only programs are not stored in the new disk file.

User action: Use the FCOPY command if you want to move read-only programs from one disk to another. Protected programs cannot be moved from one disk to another.

***Warning* SET.SPEED switch disabled (54)**

Explanation: A PRIME operation has been performed from the manual control pendant while the SET.SPEED system switch is disabled. Therefore, the monitor speed specified in the PRIME command has no effect.

User action: If you want the PRIME command to change the monitor speed, type the command **enable set.speed** at the keyboard.

***Warning* Watchdog timer disabled (56)**

Explanation: Displayed at startup by all CPUs if the watchdog timer on the board is disabled. For Adept CPUs, the timer is enabled by removing a jumper. This timer is a hardware device that asserts SYSFAIL on the VME bus (which drops high power) if the CPU halts or gets hung. On the Adept 030 board, the green light goes out if SYSFAIL is asserted.

This message also is displayed whenever user task is started from the monitor and the watchdog timer is disabled.

User action: Replace the watchdog timer jumper. See the ***Adept MV Controller User's Guide***.

Watchpoint changed at (task) program_name, step n. ... (18)

Explanation: A watchpoint has been enabled, and the watchpoint expression has changed.

User action: Continue debugging session.

***Wrong disk loaded* (–521)**

Explanation: The diskette in a disk drive has been changed while a file was still open. Further attempts to access the file result in this error. Data being written into the file may be lost.

User action: Check your diskette to see if any data was lost. If so, it's too late now. Be more careful in the future.

Numerical List

This section lists all the V+ messages that have a numeric code. Most message codes associated with errors can be made available to a program by the ERROR function, which returns the code of the latest error that occurred. In addition, the $ERROR function returns the error message associated with any V+ error code.

The information for each message below consists of the message code, the text of the message, and sometimes a comment about the applicability of the message. Angle brackets (<...>) are used to enclose a description of an item that would appear in that position. All numbers are decimal.

Table B-5, "Informational Messages" on page 764 lists messages that provide information.

Table B-6, "Warning Messages" on page 764 lists warning messages that you may receive.

Table B-7, "Error Messages" on page 766 lists the error messages that you may receive.

Table B-5. Informational Messages

Code	Message Text	Comments
0	Not complete	
1	Success	(General success response)
2	<no message>	(Signals start of program execution)
3	Program completed	
4	Program task # stopped at	
5	<no message>	(Signals start of DO processing)
6	<no message>	(Signals completion of DO processing)
7	<program instruction step>	(For TRACE mode of execution)
8	(HALTED)	
9	(PAUSED)	
10	Are you sure (Y/N)?	
11	Change?	
15	Program HOLD	
16	*Input error* Try again:	
17	Breakpoint at (task) program_name, step n	
18	Watchpoint changed at (task) program_name, step n Old value: n, New value: n	

Table B-6. Warning Messages

Code	Message Text
50	Executing in DRY.RUN mode
51	*Warning* Not calibrated
52	*Warning* Protected and read-only programs are not stored
53	*Protected program*

Table B-6. Warning Messages (Continued)

Code	**Message Text**
54	*Warning* SET.SPEED switch disabled
55	*Warning* Monitoring watchpoint
56	*Warning* Watchdog timer disabled
57	Press HIGH POWER button to enable power
58	Release then press Hold-to-run button
60	Press HIGH POWER button when blinking
100	Network connection opened
101	Network connection closed

Table B-7. Error Messages

Code	Message Text
–300	*Illegal monitor command*
–301	*No program specified*
–302	*DO not primed*
–303	*Keyswitch not set to AUTO*
–304	*Keyswitch not set to MANUAL*
–305	*Storage area format error*
–307	*Program not executable*
–308	*Program interlocked*
–309	*Program already exists*
–310	*Can't access protected or read–only program*
–311	*Invalid when program task active*
–312	*Can't start while program running*
–313	*Can't go on, use EXECUTE or PRIME*
–314	*Switch can't be enabled*
–315	*Invalid software configuration*
–316	*Software checksum error*
–317	*Keyswitch not set to NETWORK*
–318	*Program task not active*
–319	*Program task not in use*
–350	*Can't delete .PROGRAM statement*
–351	*First statement must be .PROGRAM*
–352	*Invalid in read–only mode*
–353	*No other program referenced*
–354	*Line too long*
–355	*Can't exit while lines attached*
–356	*Not found*
–357	*Recursive macros illegal*

Table B-7. Error Messages (Continued)

Code	Message Text
–358	*Cancelled*
–359	*Illegal in debug monitor mode*
–360	*Must be in debug mode*
–361	*Can't change modes while task running*
–362	*Can't execute from SEE program instruction*
–363	*Unknown editor command*
–364	*Can't create program in read–only mode*
–365	*Illegal in read–write mode*
–366	*Invalid when program on stack*
–380	*Breakpoint not allowed here*
–400	Aborted
–401	*Undefined value*
–402	*Illegal value*
–403	*Illegal assignment*
–404	*Illegal array index*
–405	*Illegal digital signal*
–406	*Undefined program or variable name*
–407	*Invalid argument*
–408	*Program argument mismatch*
–409	*Arithmetic overflow*
–410	*Negative square root*
–411	*Not enough storage area*
–412	*Branch to undefined label* Step nnn
–413	*Not enough program stack space*
–414	*Can't mix MC & program instructions*
–416	*String variable overflow*
–417	*String too short*
–418	*Illegal memory reference*

Table B-7. Error Messages (Continued)

Code	Message Text
–419	*Illegal when command program active*
–420	*Undefined value in this context*
–421	*Program not on top of stack*
–422	*Function already enabled*
–423	*Illegal operation*
–424	*Unknown keyword*
–425	*Calibration program not loaded*
–426	*Can't find calibration program file*
–450	*Can't interpret line*
–451	*Unexpected text at end of line*
–452	*Unknown instruction*
–453	*Ambiguous name*
–454	*Missing argument*
–455	*Invalid program or variable name*
–456	*Invalid number format*
–457	*Reserved word illegal*
–458	*Illegal expression syntax*
–459	*Missing parenthesis*
–460	*Missing quote mark*
–461	*Invalid format specifier*
–462	*Unknown function*
–463	*Invalid statement label*
–464	*Duplicate statement label*
–465	*Variable type mismatch*
–466	*Illegal use of belt variable*
–467	*Illegal .PROGRAM statement*
–468	*Duplicate .PROGRAM arguments*
–469	*Attempt to redefine variable type*: variable_name

Table B-7. Error Messages (Continued)

Code	Message Text
–470	*Attempt to redefine variable class*: variable_name
–471	*Misplaced declaration statement*
–472	*Control structure error* Step nnn
–473	*Control structure error*
–474	*Too many array indices*
–475	*Missing bracket*
–476	*Invalid qualifier*
–477	*Ambiguous AUTO invalid*
–500	*File already exists*
–501	*Nonexistent file*
–502	*Illegal I/O device command*
–503	*Device full*
–504	*Unexpected end of file*
–506	*File already open*
–507	*I/O communication error*
–508	*Device not ready*
–509	*Directory error*
–510	*Data checksum error*
–511	*Input block error*
–512	*File format error*
–513	*File not opened*
–514	*File or subdirectory name error*
–515	*Already attached to logical unit*
–516	*Not attached to logical unit*
–517	*I/O queue full*
–518	*Illegal I/O channel number*
–519	*Driver internal consistency error*
–520	*Invalid disk format*

Table B-7. Error Messages (Continued)

Code	Message Text
–521	*Wrong disk loaded*
–522	*Data error on device*
–523	*Bad block in disk header*
–524	*Communications overrun*
–525	*Illegal I/O redirection specified*
–526	*No data received*
–527	*Illegal user LUN specified*
–528	*Illegal record length*
–529	*Output record too long*
–530	*Protection error*
–531	*Communication time–out*
–532	*Out of I/O buffer space*
–533	*Invalid hardware configuration*
–534	*Network restarted remotely*
–535	*Network closed locally*
–536	*Too many network errors*
–537	*Unknown network node*
–538	*Network node off line*
–539	*No matching connection*
–540	*Invalid connection specified*
–541	*Invalid network protocol*
–542	*Network not enabled*
–543	*Illegal when network enabled*
–544	*Not configured as accessed*
–545	*Nonexistent subdirectory*
–546	*Subdirectory list too long*
–547	*Subdirectory in use*
–548	*Illegal while protocol active*

Table B-7. Error Messages (Continued)

Code	Message Text
–549	*Out of graphics memory*
–550	*Too many windows*
–551	*Font not loaded*
–552	*Graphics processor timeout*
–553	*Too many arguments*
–554	*Too many closeable windows*
–555	*Graphics storage area format error*
–556	*Too many icons*
–557	*Can't create new slide bar*
–558	*Graphics software checksum error*
–559	*Out of network resources*
–560	*Invalid network resource*
–561	*Invalid network address*
–562	*Network timeout*
–563	*Remote has not exported network resource*
–564	*Network resource name conflict*
–565	*Network connection terminated*
–566	*Not owner*
–567	*Not a directory*
–568	*Is a directory*
–569	*File too large*
–570	*File name too long*
–571	*Directory not empty*
–600	*Stopped due to servoing error*
–601	*Robot not attached to this program*
–602	*Robot already attached to program*
–603	*COMP mode disabled*
–604	*Robot power off*

Table B-7. Error Messages (Continued)

Code	**Message Text**
–605	*Robot not calibrated*
–606	*Joint 1 in brake track or robot overheated*
–607	*No air pressure*
–608	*External E-STOP*
–609	*Illegal joint number*
–610	*Location out of range*
–611	*Must use straight–line motion*
–612	*Straight–line motion can't alter configuration*
–613	*Illegal motion from here*
–614	*Attempt to modify active belt*
–615	*Belt not enabled*
–616	*Belt window violation*
–617	*Belt servo dead*
–618	*Location too close*
–619	*Invalid orientation*
–620	*Speed pot or STEP not pressed*
–621	*Robot interlocked*
–622	*No robot connected to system*
–623	*Stop–on–force triggered*
–624	*Force protect limit exceeded*
–625	*Invalid servo initialization data*
–626	*Can't ALTER and track belt*
–627	*Robot power on*
–628	*Robot module not loaded* ID:n
–629	*SYSFAIL asserted
–630	*Motion interface E-STOP*
–631	*Controller overheating*
–632	*Power failure detected by robot*

Table B-7. Error Messages (Continued)

Code	Message Text
–633	*PANIC command*
–635	*Cartesian control of robot not possible*
–636	*Trajectory clock overrun*
–637	*Illegal while joints SPIN'ing*
–638	*SPIN motion not permitted*
–639	*Manual brake release*
–640	*E-STOP from robot*
–641	*E-STOP from amplifier*
–642	*E-STOP from SYSFAIL*
–643	*E-STOP from backplane*
–644	*Incompatible robot and safety ID*
–645	*Power disabled: Manual/Auto changed*
–646	*HIGH POWER button on VFP not pressed*
–647	*Collision avoidance dead-lock*
–648	*User has not tested Cat3 system*
–649	*Timeout: Hold-to-run not toggled*
–650	*Manual control pendant failure*
–651	*RSC communications failure*
–652	*RSC reset*
–653	*RSC time–out*
–654	*RSC transmission garbled*
–655	*RSC bad packet format*
–656	*RSC calibration load failure*
–658	*Device hardware not present*
–659	*Device time–out*
–660	*Device error*
–661	*NVRAM data invalid*
–662	*Device sensor error*

Table B-7. Error Messages (Continued)

Code	Message Text
–663	*Device reset*
–665	*NVRAM battery failure*
–666	*Must use CPU #1*
–667	*Power failure detected*
–668	*Device in use*
–669	*RSC hardware failure*
–670	*RSC power failure*
–671	*Servo board 12V fuse open*
–672	*Servo board solenoid fuse open*
–673	*Servo board E-Stop fuse open*
–674	*Servo task overloaded*
–675	*Timeout enabling power*
–676	*RSC module ID doesn't match robot*
–701	*VISION not enabled*
–702	*No prototypes*
–703	*Too many vision requests pending*
–704	*No objects seen*
–705	*Camera not running*
–706	*Invalid request while camera running*
–707	*Unknown prototype*
–708	*Display interface absent*
–710	*Camera disconnected*
–712	*Maximum number of prototypes exceeded*
–713	*Vision not calibrated*
–714	*Camera already running*
–717	*Not enough prototype storage area*
–718	*Duplicate prototype name*
–719	*Camera already off*

Table B-7. Error Messages (Continued)

Code	Message Text
–720	*Vision option not installed*
–721	*Bad grip definition*
–722	*Camera interface board absent*
–723	*No picture data available*
–724	*Illegal display mode*
–726	*Bad camera calibration*
–727	*Invalid vision X/Y ratio*
–728	*Image processing board failure*
–729	*Invalid request while vision training*
–730	*Information not available*
–731	*Unknown sub–prototype*
–732	*Invalid model name*
–733	*Vision system out of memory*
–734	*Can't open vision window for read/write*
–735	*Invalid vision argument*
–736	*Font not defined*
–737	*Font already defined*
–738	*Font not completely trained*
–739	*Maximum number of samples trained*
–740	*Duplicate character in font*
–741	*Invalid character in font*
–742	*Character not in font*
–743	*Region too big*
–744	*Region too complicated*
–745	*Expected character(s) not found*
–746	*Template of uniform intensity*
–747	*Template not defined*
–748	*Template already defined*

Table B-7. Error Messages (Continued)

Code	Message Text
–749	*Vision aborted*
–750	*Mixing half and full resolutions*
–751	*No vision sytem selected*
–752	*AOI not defined*
–753	*Out of vision transform memory*
–754	*Correlation template too big*
–755	*Data overflow*
–756	*A scratch frame store is needed (use VSELECT)*
–757	*Inconsistent heirarchy levels*
–758	*No models*
–759	*Unknown model*
–760	*Duplicate model*
–761	*No models planned*
–800	*Unknown error code*
–801	*Invalid VFEATURE access*
–802	*Invalid camera calibration*
–803	*Illegal camera number*
–804	*Option not installed*
–805	*Hardware not in system*
–859	*Database manager internal error*
–900	*Robot module not enabled*
–901	*Obstacle collision detected*
–999	Aborted
–1001	*Invalid servo error* Mtr n
–1002	*Position out of range* Jt
–1003	*Time–out nulling errors* Mtr n
–1004	*No zero index* Mtr n
–1005	*Unexpected zero index* Mtr n

Table B-7. Error Messages (Continued)

Code	**Message Text**
–1006	*Envelope error* Mtr n
–1007	*Motor stalled* Mtr n
–1008	*Encoder quadrature error* Mtr n
–1009	*Timeout enabling amplifier* Mtr n
–1010	*Invalid error code* Belt n
–1011	*No zero index* Belt n
–1012	*Unexpected zero index* Belt n
–1013	*Encoder quadrature error* Belt n
–1014	*[Fatal] Initialization failure* Mtr n
–1015	*Initialization failure* Belt n
–1016	*Motor overheating* Mtr n
–1018	*Motor amplifier fault* Mtr n
–1021	*Duty–cycle exceeded* Mtr n
–1022	*Skew envelope error* Mtr n
–1023	*Position out of range* Mtr n
–1025	*Encoder fault*
–1026	*Software incompatible* Code *n*
–1027	*Hard envelope error*
–1032	*Negative overtravel* Mtr n
–1033	*Positive overtravel* Mtr n
–1034	*Overtravel* Mtr n
–1101	*[Fatal] Servo process dead* CPU n
–1102	*[Fatal] Servo code incompatible* CPU n
–1104	*[Fatal] Servo dead* Mtr n
–1105	*Motor startup failure* Mtr n
–1106	*Calibration sensor failure* Mtr n
–1107	*[Fatal] Servo init failure* CPU n
–1108	*Cat3 diagnostic error* Code n

Table B-7. Error Messages (Continued)

Code	**Message Text**
–1109	*Cat3 external sensor fault* Code n
–1111	*Cat3 external E-STOP* Code n
–1200 to –1299	*NFS error* Code *n**

C ID Option Words

Introduction

This appendix supplements the descriptions of the ID Monitor Command in the ***V+Operating System Reference Guide*** and the ID( ) Real-Valued Function in the ***V+Language Reference Guide***.

The ID command displays various option words as hexadecimal values; the ID function makes the same values available to programs. This appendix describes the following:

- Basic V+ system (two option words)
- Controller option word
- Processor (option word for each processor)
- Vision option word (option word for each vision interface)

System Option Words

The configuration of a specific V+ system can be determined by examining two "option words" that are displayed (as hexadecimal numbers) when the system is loaded from the system boot disk, and by the ID monitor command. The values of the option words are also available to programs from the real-valued functions ID(5) and ID(6).

The option words should be interpreted as collections of bit fields, each of which indicates information about the system configuration. The interpretations of the bits in the first system option word [returned by the function ID(5)] are described in **Table C-1, "System Option Word #1 [from ID(5)]" on page 781**.

Table C-1. System Option Word #1 [from ID(5)]

Bit #	Mask Value		Interpretation When Bit Set
	Decimal	**Hexadecimal**	
1	1	1	"V+ Extensions" software license installed
2	2	2	External encoders are supported[a]
3-7			Reserved for future use (currently zero)
8	128	80	Alter instruction enabled[b]

[a] The External encoder bit is used with robot systems to indicate the "conveyor tracking" capability. With nonrobot systems it is used to indicate the "external encoder option".
[b] This bit tracks the "V+ Extensions" bit.

The interpretations of the bits in the second software option word [returned by the function ID(6)] are described in **Table C-2**.

Table C-2. System Option Word #2 [from ID(6)]

Bit #	Mask Value		Interpretation When Bit Set
	Decimal	**Hexadecimal**	
1-6			Reserved for future use (currently zero)
7	64	40	Guidance vision is enabled
8	128	80	Inspection vision is enabled
9	256	100	DDCMP option is installed[a]
10-11			Reserved for future use (currently zero)
12			AdeptNet hardware is installed, and TCP and or NFS is installed. You can use the NETWORK real-valued function to obtain detailed information about the AdeptNet option.
13-16			Reserved for future use (currently zero)

[a] This bit tracks the "V+ Extensions" bit in the first option word.

Controller Option Word

This word is not used, and all bits currently are zero.

> **NOTE:** The installed Software Licenses can be listed using the CONFIG_C utility.

Robot Option Words

For information about the first robot option word, consult the documentation for your particular kinematic module.

The real-valued functions ID(11,8) and ID(11,10+`robot`) return the second option word for the selected and specified robot, respectively. The interpretations of the bits in this option word are described in **Table C-3**.

Table C-3. Robot Option Word #2 (from ID(11, 10+*robot*)]

Bit #	Mask Value		Interpretation When Bit Set
	Decimal	**Hexadecimal**	
1	1	1	Robot has an RSC.
2	2	2	Robot has an extended-length quill.
3	4	4	Robot has the cleanroom option.
4	8	8	Robot has the HyperDrive option. This bit also enables the MOVEF and MOVESF program instructions.
5	16	10	Robot has the high-torque option.
6			Reserved for future use (currently zero).
7	64	40	Robot has the EC certification option.
8–16			Reserved for future use (currently zero).

Processor Option Word

The interpretations of the bits in the processor option word [returned by the function ID(6, 4)] are described in **Table C-4**.

Table C-4. Processor Option Word [from ID(6, 4)]

Bit #	Mask Value		Interpretation When Bit Set
	Decimal	**Hexadecimal**	
1	1	1	Processor is running the V+ Operating System
2	2	2	Processor is running the Vision processing software
3	4	4	Processor is running the Servo software
4-16			Reserved for future use (currently zero)

Vision Option Word

There is one vision option word for each vision interface (maximum of two).

The interpretations of the bits in the vision option word [returned by the function ID(5, 3)] are described in **Table C-5**.

Table C-5. Vision Option Word [from ID(5, 3)]

Bit #	Mask Value		Interpretation When Bit Set
	Decimal	**Hexadecimal**	
1-4	15	F	CPU number for this interface module
5			Reserved for future use (currently zero)
6	32	20	Extended Vision Interface (EVI) present
7-8			Reserved for future use (currently zero)
9-10	768	300	Reserved for internal use by Adept (may be one or zero)
11-16			Reserved for future use (currently zero)

Glossary D

The following terms may be new to you, or their use in this manual may differ from your previous experience. Some of these terms are explained more fully elsewhere in this manual.

Array	A collection of stored values (locations, real values, or strings) that can be referenced individually or collectively. A symbolic name is used to refer to an array. The individual values are called array elements and are referenced by appending an index to the array name. For example, a group of robot locations can be stored as an array part and the individual locations can be referred to as part[0], part[1], part[2], and so on.
ASCII	The acronym for American Standard Code for Information Interchange, which is the name of a system for assigning numeric values to the characters used by a computer.
Binary	The name of the base-two number system. That is, the number system that uses only the digits 0 and 1. (See Decimal, Hexadecimal, and Octal.)
Breakpoint	A flag in a program that causes the program to stop executing and/or display a value of interest to the programmer while debugging the program. (See Debugging and Watchpoint.)
Command	A directive from the user to the V+ monitor, to the V+ SEE editor, or to the V+ program debugger.
Compound transformation	A transformation that is defined as a combination of relative transformation values, joined with colons (n:n).
Debugging	The process of executing programs interactively to detect and correct program errors. (See Breakpoint and Watchpoint.)

Decimal	The name of the base-ten number system. That is, the common number system that uses only the digits 0 to 9. Unless otherwise noted, all numbers in this manual are decimal. (Also see Binary, Hexadecimal, and Octal.)
Default	When an optional argument is omitted from a V+ command or instruction, some value is assumed for the argument. The value assumed is referred to as a default value.
Editor	An aid for entering information into a computer system and modifying existing text. The V+ editors are used to enter and modify application programs.
Expression	A combination of real-valued variables and functions, and mathematical and logical operators that, when evaluated, yields a numeric value.
File	A collection of information stored on a device that is peripheral to the V+ system controller memory. For example, V+ application programs can be stored in files in the system disk drive.
Function	A V+ language element that results in a value being returned in its place.
Hexadecimal	The name of the base-sixteen number system. That is, the number system that uses 0 to 9 and A to F as its digits. (Also see Binary, Decimal, and Octal.) Hexadecimal numbers are frequently convenient to use because their conversion to binary (base-two) numbers is very simple--one scans the hexadecimal number from left to right converting each digit to its 4-bit binary equivalent. For example: 123ABC (hex) = 0001 0010 0011 1010 1011 1100 (binary)
Instruction	A directive to the V+ system that can be recorded in a program.
Integer	A numeric value that does not have a fractional part. In the V+ system, integers can range from –16,777,216 to +16,777,215 without losing any precision. (That is, larger values may be truncated to seven significant digits.)

Kilobyte	A unit of measure (abbreviated K) used when describing the amount of information that can be stored in a system component (for example, memory). A byte is an 8-bit storage element, which can hold a single character or a portion of a numeric value. The abbreviation K is used to represent a multiple of 1024. Thus, for example, 256K represents 256 * 1024 (262,144) bytes. (Also see Megabyte.)
Keyword	A word that has a predefined meaning to V+. For example, all the command, instruction, and function names are keywords. Those keywords that cannot be used as program or variable names are called reserved keywords.
Label	A number used to identify a V+ program instruction for the purpose of having program execution branch to that instruction. (This should not be confused with a Step Number.)
Location	The description of the position of an object in space and the orientation of the object. Locations are used to define the positions and orientations the robot tool is to assume during program execution. (See Point.)
Logical value	A numeric value that is interpreted as either true or false. A zero value is interpreted by V+ as false, and a nonzero value is interpreted as true.
Megabyte	A unit of measure (abbreviated MB) used when describing the amount of information that can be stored in a system component (for example, a disk). A byte is an 8-bit storage element, which can hold a single character or a portion of a numeric value. The abbreviation M is used to represent a multiple of 1024 * 1024 (1,048,576). Thus, for example, 10 MB represents 10 * 1,048,576 (10,485,760) bytes. (Also see Kilobyte.)
Monitor	An administrative computer program that oversees operation of a system. The V+ monitor accepts user input and initiates the appropriate response; follows instructions from application programs to direct the robot; and performs the computations necessary to control the robot.

Octal	The name of the base-eight number system. That is, the number system that uses only the digits 0 to 7. (Also see Binary, Decimal, and Hexadecimal.) Octal numbers are frequently convenient to use because their conversion to binary (base-two) numbers is very simple--one scans the octal number from left to right converting each digit to its 3-bit binary equivalent. For example: 1234 (octal) = 001 010 011 100 (binary)
Operation	A general term used in this manual to refer to V^+ commands, instructions, and functions.
Operator	An indicator of a mathematical, relational or logical operation to be performed. For example, + is the addition operator in V^+. OR is the logical-or operator.
Parameter	A numeric variable that determines characteristics of the operation of the V^+ system. (Also see "Switch".)
Point	A position in space defined, for example, by its X, Y, and Z coordinate values. Unlike a location (see above), a point does not represent any orientation information. Thus, a location represents a point and an orientation at that point.
Precision point	A description of a location that specifies the position of each of the robot joints. (Also see Transformation.)
Program	A list of instructions telling a computer how to do a desired task. For example, V^+ programs are written to describe tasks the robot is to perform.
Real value	A numeric value that can have an integer part and a fractional part. For example, 15.25 has an integer part equal to 15 and a fractional part equal to 0.25.
Relative transformation	A transformation that defines a location relative to another location, rather than relative to the origin of the robot coordinate system.
Scalar	A single value, as opposed to an array of values. (See Array.)
Switch	A logical variable that determines characteristics of the operation of the V^+ system. That is, a switch can be set to either of two states, which are referred to as enabled and disabled. (Also see Parameter.)

Transformation	A mathematical description of a location, which defines the position and orientation of the location without regard for the configuration of the robot when it is at the location. (Also see Precision point.) The term "transform" is sometimes used to refer to a transformation.
Variable	A stored value that is referred to with a symbolic name. For example, a robot location can be referred to as start, and a real value can be called loop.count.
Watchpoint	A variable or expression that is evaluated before each step of a program is executed, and that causes program execution to stop if the value has changed. (Also see Breakpoint and Debugging.)
VFI	The VME Force Interface module.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Adept User's Manual Comment Form

We have provided this form to allow you to make comments about this manual, to point out any mistakes you may find, or to offer suggestions about information you want to see added to the manual. We review and revise user's manuals on a regular basis, and any comments or feedback you send us will be given serious consideration. Thank you for your input.

NAME_____DATE_____

COMPANY _____

ADDRESS _____

PHONE _____

MANUAL TITLE: *V+ Language Reference Guide*
PART NUMBER: 00962-01100 PUBLICATION DATE: September, 1997

COMMENTS _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

MAIL TO: Adept Technology, Inc.
Technical Publications Dept.
11133 Kenwood Rd.
Cincinnati, OH 45242